United States Environmental Protection Agency

Printed April 2008

User's Manual
TEVA-SPOT Toolkit 2.2

by
Jonathan Berry, Erik Boman, Lee Ann Riesen
Scalable Algorithms Dept
Sandia National Laboratories
Albuquerque, NM 87185

William E. Hart, Cynthia A. Phillips, Jean-Paul Watson
Discrete Math and Complex Systems Dept
Sandia National Laboratories
Albuquerque, NM 87185

Project Officer:
Regan Murray
National Homeland Security Research Center
Office of Research and Development
U.S. Environmental Protection Agency
Cincinnati, OH 45256

The U.S. Environmental Protection Agency (EPA) through its Office of Research and Development funded and collaborated in the research described here under an Inter-Agency Agreement with the Deparment of Energy's Sandia National Laboratories (IAG # DW8992192801). This document has been subjected to the Agency's review, and has been approved for publication as an EPA document. EPA does not endorese the purchase or sale of any commercial products or services.

Questions concerning this document or its application should be addressed to:

Regan Murray
USEPA/NHSRC (NG 16)
26 W Martin Luther King Drive
Cincinnnati OH 45268
(513) 569-7031
Murray.Regan@epa.gov

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under Contract DE-AC04-94-AL85000.

Forward

Since its inception in 1970, EPA's mission has been to pursue a cleaner, healthier environment for the American people. The Agency was assigned the daunting task of repairing the damage already done to the natural environment and establishing new criteria to guide Americans in making a cleaner environment a reality. Since 1970, the EPA has worked with federal, state, tribal, and local partners to advance its mission to protect human health and the environment. In order to carry out its mission, EPA employs and collaborates with some of the nation's best scientific minds. EPA prides itself in applying sound science and state of the art techniques and methods to develop and test innovations that will protect both human health and the environment.

Under existing laws and recent Homeland Security Presidential Directives, EPA has been called upon to play a vital role in helping to secure the nation against foreign and domestic enemies. The National Homeland Security Research Center (NHSRC) was formed in 2002 to conduct research in support of EPA's role in homeland security. NHSRC research efforts focus on five areas: water infrastructure protection, threat and consequence assessment, decontamination and consequence management, response capability enhancement, and homeland security technology testing and evaluation. EPA is the lead federal agency for drinking water and wastewater systems and the NHSRC is working to reduce system vulnerabilities, prevent and prepare for terrorist attacks, minimize public health impacts and infrastructure damage, and enhance recovery efforts.

This Users Manual for the TEVA-SPOT Toolkit software package is published and made available by EPA's Office of Research and Development to assist the user community and to link researchers with their clients.

Jonathan Herrmann, Director

National Homeland Security Research Center
Office of Research and Development
U. S. Environmental Protection Agency

License Notice

Acknowledgements

The National Homeland Security Research Center would like to acknowledge the following organizations and individuals for their support in the development of the TEVA-SPOT Toolkit User's Manual and/or in the development and testing of the TEVA-SPOT Toolkit Software.

Office of Research and Development - National Homeland Security Research Center

Robert Janke
Regan Murray
Terra Haxton

Sandia National Laboratories

Jonathan Berry
Erik Boman
William Hart
Lee Ann Riesen
Cynthia Phillips
Jean-Paul Watson

Argonne National Laboratory

Thomas Taxon

University of Cincinnati

James Uber

American Water Works Association Utility Users Group

Kevin Morley (AWWA)

Contents

1 Introduction

Public water distribution systems are inherently vulnerable to accidental or intentional contamination because of their distributed geography. Further, there are many challenges to detecting contaminants in drinking water systems: municipal distribution systems are large, consisting of hundreds or thousands of miles of pipe; flow patterns are driven by time-varying demands placed on the system by customers; and distribution systems are looped, resulting in mixing and dilution of contaminants. The use of on-line, real-time contaminant warning systems (CWSs) is a promising strategy for mitigating these risks. Online sensor data can be combined with public health surveillance systems, physical security monitoring, customer complaint surveillance, and routine sampling programs to effect a rapid response to contamination incidents [20].

A variety of technical challenges need to be addressed to make CWSs a practical, reliable element of water security systems. A key aspect of CWS design is the strategic placement of sensors throughout the distribution network. Given a limited number of sensors, a desirable sensor placement minimizes the potential economic and public health impacts of a contaminant incident. There are a wide range of important design objectives for sensor placements (e.g., minimizing the cost of sensor installation and maintenance, the response time to a contamination incident, and the extent of contamination). In addition, flexible sensor placement tools are needed to analyze CWS designs in large scale networks.

1.1 What is TEVA-SPOT?

The Threat Ensemble Vulnerability Assessment and Sensor Placement Optimization Tool (TEVA-SPOT) has been developed by the U. S. Environmental Protection Agency, Sandia National Laboratories, Argonne National Laboratory, and the University of Cincinnati. TEVA-SPOT has been used to develop sensor network designs for several large water utilities [11], including the pilot study for the EPA's Water Security Initiative.

TEVA-SPOT allows a user to specify a wide range of modeling inputs and performance objectives for contamination warning system design. Further, TEVA-SPOT supports a flexible decision framework for sensor placement that involves two major steps: a modeling process and a decision-making process [12]. The modeling process includes (1) describing sensor characteristics, (2) defining the design basis threat, (3) selecting impact measures for the CWS, (4) planning utility response to sensor detection, and (5) identifying feasible sensor locations.

The design basis threat for a CWS is the ensemble of contamination incidents that a CWS should be designed to protect against. In the simplest case, a design basis threat is a contamination scenario with a single contaminant that is introduced at a specific time and place. Thus, a design basis threat consists of a set of contamination incidents that can be simulated with standard water distribution models [17]. TEVA-SPOT provides a convenient interface for defining and computing the impacts of design basis threats. In particular, TEVA-SPOT can simulate many contamination incidents in parallel, which has reduced the computation of very large design basis threats from weeks to hours on the EPAs high performance computing system.

TEVA-SPOT was designed to model a wide range of sensor placement problems. For example, TEVA-SPOT supports a number of impact measures, including the number of people exposed to dangerous levels of a contaminant, the volume of contaminated water used by customers, the number of feet of contaminated pipe, and the time to detection. Response delays can also be specified to account for the time a water utility would need to verify a contamination incident before notifying the public. Finally, the user can specify the feasible locations for sensors and fix sensor locations during optimization. This flexibility allows a user to evaluate how different factors impact the CWS performance and to iteratively refine a CWS design.

1.2 About This Manual

The capabilities of TEVA-SPOT can be accessed either with a GUI or from command-line tools. This user manual describes the TEVA-SPOT Toolkit, which contains these command-line tools. The TEVA-SPOT Toolkit can be used within either a MS Windows DOS shell or any standard Unix shell (e.g. the Bash shell).

The following sections describe the TEVA-SPOT Toolkit, which we refer to as SPOT throughout this manual:

- **TEVA-SPOT Toolkit Basics** - An introduction to the process of sensor placement, the use of SPOT command-line tools, and installation of the SPOT executables.
- **Sensor Placement Formulations** - The mathematical formulations used by the SPOT solvers.
- **Contamination Incidents and Impact Measures** - A description of how contamination incidents are computed, and the impact measures that can be used in SPOT to analyze them.
- **Sensor Placement Solvers** - A description of how to apply the SPOT sensor placement solvers.
- **File Formats** - Descriptions of the formats of files used by the SPOT solvers.

In addition, the appendices of this manual describe the syntax and usage of the SPOT command-line executables.

2 TEVA-SPOT Toolkit Basics

This section provides an introduction to the process of sensor placement, the use of SPOT command-line tools, and the installation of the SPOT executables.

2.1 Approaches to Sensor Placement

Sensor placement strategies can be broadly characterized by the technical approach and the type of computational model used. The following categories reflect important differences in proposed sensor placement strategies:

- **Expert Opinion:** Although expertise with water distribution systems is always needed to design an effective CWS, here we refer to approaches that are solely guided by expert judgement. For example, Berry et al. [4] and Trachman [19] consider sensor placements developed by experts with significant knowledge of water distribution systems. These experts did not use computational models to carefully analyze network dynamics. Instead, they used their experience to identify locations whose water quality is representative of water throughout the network.

- **Ranking Methods:** A related approach is to use preference information to rank network locations [1, 8]. In this approach, a user provides preference values for the properties of a "desirable" sensor location, such as proximity to critical facilities. These preferences can then be used to rank the desirability of sensor locations throughout the network. Further, spatial information can be integrated to ensure good coverage of the network.

- **Optimization:** Sensor placement can be automated with optimization methods that computationally search for a sensor configuration that minimizes contamination risks. Optimization methods use a computational model to estimate the performance of a sensor configuration. For example, a model might compute the expected impact of an ensemble of contamination incidents, given sensors placed at strategic locations.

  Optimization methods can be further distinguished by the type of computational model that they use. Early sensor placement research focused on models that used simplified network models derived from contaminant transport simulations. For example, hydraulic simulations can be used to model stable network flows [3], or to generate an *averaged* water network flow model [14].

  More recently, researchers have used models that directly rely on contaminant transport simulation results. Simulation tools, like EPANET [17], perform extended-period simulation of the hydraulic and water quality behavior within pressurized pipe networks. These models can evaluate the expected flow in water distribution systems, and they can model the transport of contaminants and related chemical interactions. Thus, the CWS design process can directly minimize contamination risks by considering simulations of an ensemble of contamination incidents, which reflect the impact of contamination at different locations, times of the day, etc.

SPOT development has focused on optimization methods, and in particular on methods that use contaminant transport simulation. Contaminant transport simulation models can directly model contamination risks, and consequently optimization methods using these models have proven effective at minimizing risk. Comparisons with expert opinion and ranking methods suggest that these approaches are not as effective in large, complex networks [4, 15]. Further, optimization methods using simpler models can fail to capture important transient dynamics (see Berry et al. [6] for a comparison).

A key issue for the simulation-based optimization methods is that they require the simulation of a potentially large number of contamination incidents. Consequently, it is very expensive to apply generic optimization

methods like evolutionary algorithms [14]. However, Berry et al. [5] have shown that these simulations can be performed in an off-line preprocessing step that is done in advance of the optimization process. Thus, the time needed for simulation does not necessarily impact the time spent performing sensor placement.

2.2 The Main Steps in Using SPOT

The following example illustrates the main steps required to (1) simulate contamination incidents, (2) compute contamination impacts, (3) perform sensor placement, and (4) evaluate a sensor placement. This example places sensors in EPANET Example 3 (Net3), a small distribution system with 97 junctions.

The data used in this example is available in the C:\spot\examples\simple directory. In general, a user will need to use a variety of data sources to develop a sensor placement model. The file C:\spot\doc\SPOT_-DataRequirements.doc discussess the type of data used in TEVA-SPOT in greater detail.

2.2.1 Simulating Contamination Incidents

Simulation of contamination incidents is performed with the `tevasim` command, which iteratively calls EPANET to simulation an ensemble of contamination incidents. The `tevasim` command has the following inputs and outputs:

- **Inputs:**
  - TSG File: defines an ensemble of contamination scenarios
  - INP File: the EPANET input file for the network
- **Outputs:**
  - TSO File: a binary file that stores the contamination results for all incidents
  - SDX File: a binary index file into the TSO file
  - OUT File: a plain text log file

For example, the file C:\spot\examples\simple\Net3.tsg defines an ensemble of contamination scenarios for Net3. Contamination incidents are simulated for all network junctions, one for each hour of the day, and each contamination incident models an inject that continues for 24 hours. The `tevasim` command performs these contaminant transport simulations, using the following command line:

```
tevasim --tsg Net3.tsg --tso Net3.tso Net3.inp Net3.out
```

2.2.2 Computing Contamination Impacts

A TSO file contains raw data about the extent of a contamination throughout a network. This data needs to be post-processed to compute relevant impact statistics. The `tso2Impact` command processes a TSO file and generates one or more IMPACT files. An `IMPACT` file is a plain text file that summarizes the consequence of each contamination incident in a manner that facilitates optimization. The `tso2Impact` command has the following inputs and outputs:

- **Inputs:**
  - TSO File: a binary file of contamination result data generated by `tevasim`
  - SDX File: a binary index file generated by `tevasim`

  - INP File: the EPANET input file for the network, which is used to compute impact measures like the extent of contamination

- **Outputs:**
  - IMPACT File(s): plain text files that summarize the observed impact at each location where a contamination incident could be observed by a potential sensor.
  - NODEMAP File(s): plain text files that map sensor placement ids to the network junction labels (defined by EPANET).

The `tso2Impact` command generates IMPACT files with the following command line:

```
tso2Impact --mc --vc --td --nfd --ec Net3 Net3.tso
```

This command generates IMPACT files for each of the five objectives specified: mass consumed (mc), volume consumed (vc), time to detection (td), number of failed detections (nfd) and extent of contamination (ec). For each impact file (e.g. `Net3_mc.impact`), a corresponding id file is generated (e.g. `Net3_mc.impact.id`).

2.2.3 Performing Sensor Placement

An IMPACT file can be used to define a sensor placement optimization problem. The standard problem supported by SPOT is to minimize the expected impact over an ensemble of incidents while limiting the number of potential sensors. By default, sensors can be placed at any junction in the network. The `sp` command coordinates the application of optimization solvers for sensor placement. The `sp` command has a rich interface, but the simplest use of it requires the following inputs and outputs:

- **Inputs:**
  - IMPACT File(s): plain text files that summarize the observed impact at each location
  - NODEMAP File(s): plain text files that map sensor placement ids to the network junction labels
- **Outputs:**
  - SENSORS File: a plain text file that summarizes the sensor locations identified by the optimizer

For example, the command

```
sp --print-log --network="Net3" --objective=mc --solver=snl_grasp \
        --ub=ns,5 --seed=1234567
```

generates the file `Net3.sensors`, and prints a summary of the impacts for this sensor placement.

2.2.4 Evaluating a Sensor Placement

The final output provided by the `sp` command is actually generated by the `evalsensor` command, and this command can be directly used to evaluate a sensor placement for a wide variety of different objectives. The `evalsensor` command requires the following inputs:

- **Inputs:**
  -

- IMPACT File(s): plain text files that summarize the observed impact at each location
- NODEMAP File(s): plain text files that map sensor placement ids to the network junction labels
- SENSORS File: a plain text file that defines a sensor placement

For example, the command

```
evalsensor --nodemap=Net3.nodemap Net3.sensors Net3_ec.impact \
           Net3_mc.impact Net3_nfd.impact
```

will summarize the solution in the `Net3.sensors` file for the `ec`, `mc` and `nfd` impact measures. No files are generated by `evalsensors`.

2.3 Installation and Requirements for Using SPOT

Instructions for installing SPOT in Unix are included in the first appendix. Installation on MS Windows platforms is considerably easier. An installer executable can be downloaded from

```
http://www.epa.gov/nhsrc/water/teva.html
```

When run, this installer places the SPOT software in the directory

```
C:\SPOT
```

Additionally, the installer places command wrappers in the system folder, so the system path does not need to be edited to use the SPOT tools.

Some of the SPOT commands use the Python scripting language. Python is not commonly installed in MS Windows machines, but an installer script can be downloaded from

```
http://www.python.org/download/
```

Unfortunately, the system path needs to be modified to include the Python executable. A nice video describing how to edit the system path is available at:

```
http://showmedo.com/videos/video?name=960000&fromSeriesID=96
```

No other utilities need to be installed to run the SPOT commands. EPANET is linked into the `tevasim` executable. Detailed information about the SPOT commands is provided in the appendices. Note that all SPOT commands need to be run from the DOS command shell. This can be launched from the "Accessories/Command Prompt" menu. Numerous online tutorials can provide information about DOS commands. For example, see

```
http://en.wikipedia.org/wiki/List_of_DOS_commands
http://www.computerhope.com/msdos.htm
```

Note that the plain text input files used by SPOT can be edited using standard text editors. For example, at a DOS prompt you can type

```
notepad Net3.tsg
```

to open up the `Net3.tsg` file with the MS Windows Notepad application. The plain text output files can be viewed in a similar manner. The binary files generated by SPOT cannot be viewed in this manner. Generally, output files should not be modified manually since many are used as input to other programs.

2.4 Reporting Bugs and Feature Requests

The TEVA-SPOT development team uses Trac tickets to communicate requests for features and bug fixes. The TEVA-SPOT Trac site can can be accessed at:

```
https://software.sandia.gov/trac/spot
```

External users can insert a ticket, which will be moderated by the developers. Note that this is the only mechanism for ensuring that bug fixes will be made a high priority by the development team.

3 Sensor Placement Formulations

SPOT integrates solvers for sensor placement that have been developed by Sandia National Laboratories and the Environmental Protection Agency, along with a variety of academic collaborators [3, 5, 7, 9, 12, 13]. SPOT includes (1) general-purpose heuristic solvers that consistently locate optimal solutions in minutes, (2) integer- and linear-programming heuristics that find solutions of provable quality, (3) exact solvers that find globally optimal solutions, and (4) bounding techniques that can evaluate solution optimality. These solvers optimize a representation of the sensor placement problem that may be either an implicit or explicit. However, in either case we can describe the mathematical formulation for this problem.

This section describes the mixed integer programming (MIP) formulations optimized by the SPOT solvers, and this presentation assumes that the reader is familiar with MIP models. First, we describe the standard SPOT formulation, eSP, which minimizes expected impact given a sensor budget. Subsequently, we describe several other sensor placement formulations that SPOT solvers can optimize. This discussion is limited to a description of the mathematical structure of these sensor placement problems. In many cases, SPOT has more than one optimizer that can optimize these formulations, and we describe these optimizers later in this manual. However, the goal of this section is to describe the mathematical structure of these formulations.

3.1 The Standard SPOT Formulation

The most widely studied sensor placement formulation for CWS design is to minimize the expected impact of an ensemble of contamination incidents given a sensor budget. This formulation has also become the standard formulation in SPOT, since it can be effectively used to select sensor placements in large water distribution networks.

A MIP formulation for expected-impact sensor placement is:

$$
\begin{array}{llll}
\text{(eSP)} & \min & \sum_{a\in\mathcal{A}} \alpha_a \sum_{i\in\mathcal{L}_a} d_{ai} x_{ai} & \\
& \text{s.t.} & \sum_{i\in\mathcal{L}_a} x_{ai} = 1 & \forall a \in \mathcal{A} \\
& & x_{ai} \leq s_i & \forall a \in \mathcal{A}, i \in \mathcal{L}_a \\
& & x_{ai} \leq 1 - s_i & \forall a \in \mathcal{A}, i \in \mathcal{L}_a \setminus \{q\} \\
& & \sum_{i\in L} c_i s_i \leq p & \\
& & s_i \in \{0,1\} & \forall i \in L \\
& & 0 \leq x_{ai} \leq 1 & \forall a \in \mathcal{A}, i \in \mathcal{L}_a
\end{array}
$$

This MIP minimizes the expected impact of a set of contamination incidents defined by $\mathcal{A}$. For each incident $a \in \mathcal{A}$, α_a is the weight of incident a, frequently a probability. This formulation integrates contamination simulation results, which are reported at a set of *locations*, denoted L, where a location refers to a network junction. For each incident a, $\mathcal{L}_a \subseteq L$ is the set of locations that can be contaminated by a. Thus, a sensor at a location $i \in \mathcal{L}_a$ can detect contamination from incident a at the time contamination first arrives at location i. Each incident is *witnessed* by the first sensor to see it. For each incident $a \in \mathcal{A}$ and location $i \in \mathcal{L}_a$, d_{ai} defines the impact of the contamination incident a if it is witnessed by location i. This impact measure assumes that as soon as a sensor witnesses contamination, then any further contamination impacts are mitigated (perhaps after a suitable delay that accounts for the response time of the water utility). The s_i variables indicate where sensors are placed in the network; the c_i is the cost of placing a sensor at location i, and p is the budget.

The x_{ia} variables indicate whether incident a is witnessed by a sensor at location i. We may not be able to witness all contamination incidents with a given set of sensors. To account for this, L contains a *dummy* location, q. This dummy location is in all subsets $\mathcal{L}_a$. The impact for this location is handled in two different ways: (1) it is the impact of the contamination incident after the entire contaminant transport simulation

has finished, which corresponds to the impact that would occur without an online CWS, or (2) it has zero impact. The first approach treats detection by this dummy location as a penalty. The second approach simply ignores the detection by this dummy, though this does not really make sense without additional side-constraints on the number of failed detections.

The eSP formulation is a slight generalization of the sensor placement model described by Berry et al. [5]. Berry et al. treat the impact of the dummy is treated as a penalty, in which case the third constraint is redundant. The impact of a dummy detection is larger than all other impacts for each incident, so the witness variable x_{ai} for the dummy will only be selected if no sensors have been placed that can detect this incident.

Ignoring the constraint in this case, Berry et al. note that eSP is identical to the well-known p-median facility location problem [10] when $c_i = 1$. In the p-median problem, p facilities (e.g., central warehouses) are to be located on m potential sites such that the sum of distances d_{ai} between each of n customers (e.g., retail outlets) and the nearest facility i is minimized. In comparing eSP and p-median problems, we observe equivalence between (1) sensors and facilities, (2) contamination incidents and customers, and (3) contamination impacts and distances. While eSP allows placement of *at most* p sensors, p-median formulations generally enforce placement of all p facilities; in practice, the distinction is irrelevant unless p approaches the number of possible locations.

3.2 Robust SPOT Formulations

The eSP model can be viewed as optimizing one particular statistic of the distribution of impacts defined by the contaminant transport simulations. However, other statistics may provide more "robust" solutions, that are less sensitive to changes in this distribution [22]. Consider the following reformulation of eSP:

$$
\begin{array}{llll}
\text{(rSP)} & \min & Impact_f(\alpha, d, x) & \\
& \text{s.t.} & \sum_{i \in \mathcal{L}_a} x_{ai} = 1 & \forall a \in \mathcal{A} \\
& & x_{ai} \leq s_i & \forall a \in \mathcal{A}, i \in \mathcal{L}_a \\
& & x_{ai} \leq 1 - s_i & \forall a \in \mathcal{A}, i \in \mathcal{L}_a \setminus \{q\} \\
& & \sum_{i \in L} c_i s_i \leq p & \\
& & s_i \in \{0,1\} & \forall i \in L \\
& & 0 \leq x_{ai} \leq 1 & \forall a \in \mathcal{A}, i \in \mathcal{L}_a
\end{array}
$$

The function $Impact_f(\alpha, d, x)$ computes a statistic of the impact distribution. The following functions are supported in SPOT (see Watson, Hart and Murray [22] for further discussion of these statistics):

- **Mean:** This is the statistic used in eSP.
- **VaR:** Value-at-Risk (VaR) is a percentile-based metric. Given a confidence level $\beta \in (0,1)$, the VaR is the value of the distribution at the $1-\beta$ percentile [18]. The value of VaR is less than the TCE value.

  Mathematically, suppose we have a random variable W that describes the distribution of possible impacts. Then we have

  $$\text{VaR}(W, \beta) = \min\{w \mid \Pr[W \leq w] \geq \beta\}.$$

  Note that the distribution W changes with each sensor placement. Further, VaR can be computed using the α, d and x values.
- **TCE:** The Tail-Conditioned Expectation (TCE) is a related metric which measures the conditional expectation of impact exceeding VaR at a given confidence level. Given a confidence level $1-\beta$, TCE is the expectation of the worst impacts with probability β. This value is between VaR and the worst-case value.

Mathematically, we have

$$\mathrm{TCE}(\beta) = \mathrm{E}\left[W \mid W \geq \mathrm{VaR}(\beta)\right].$$

The Conditional Value-at-Risk (CVaR) is a linearization of TCE investigated by Uryasev and Rockafellar [16]. CVaR approximates TCE with a continuous, piecewise-linear function of β, which enables the use of CVaR in a MIP models for rSP.

- **Worst:** The worst impact value can be easily computed, since a finite number of contamination incidents are simulated. Further, rSP can be reworked to formulate a worst-case MIP formulation. However, this statistic is sensitive to changes in the number of contamination incidents that are modeled; adding additional contamination incidents may significantly impact this statistic.

3.3 Min-Cost Formulations

A standard variant of eSP and rSP is to minimize cost while constraining the impact to be below a specified threshold, u . For example, the eSP MIP can be revised to formulate a MIP to minimize cost:

$$\begin{array}{lll}
\text{(ceSP)} \quad \min & \sum_{i \in L} c_i s_i & \\
\text{s.t.} & \sum_{i \in \mathcal{L}_a} x_{ai} = 1 & \forall a \in \mathcal{A} \\
& x_{ai} \leq s_i & \forall a \in \mathcal{A}, i \in \mathcal{L}_a \\
& x_{ai} \leq 1 - s_i & \forall a \in \mathcal{A}, i \in \mathcal{L}_a \setminus \{q\} \\
& \sum_{a \in \mathcal{A}} \alpha_a \sum_{i \in \mathcal{L}_a} d_{ai} x_{ai} \leq u & \\
& s_i \in \{0,1\} & \forall i \in L \\
& 0 \leq x_{ai} \leq 1 & \forall a \in \mathcal{A}, i \in \mathcal{L}_a
\end{array}$$

Minimal cost variants of rSP can also be easily formulated.

3.4 Formulations with Multiple Objectives

CWS design generally requires the evaluation and optimization of a variety of performance objectives. Some performance objectives cannot be simultaneously optimized, and thus a CWS design must be selected from a trade-off between these objectives [21].

SPOT supports the analysis of these trade-offs with the specification of additional constraints on impact measures. For example, a user can minimize the expected extent of contamination (ec) while constraining the worst-case time to detection (td). SPOT allows for the specification of more than one impact constraint. However, the SPOT solvers cannot reliably optimize formulations with more than one impact constraint.

3.5 The SPOT Formulation with Imperfect Sensors

The previous sensor placement formulations make the implicit assumption that sensors work perfectly. That is, they never fail to detect a contaminant when it exists, and they never generate an erroneous detection when no contaminant exists. In practice, sensors are imperfect, and they generate these types of errors.

SPOT addresses this issue by supporting a formulation that models simple sensor failures [2]. Each sensor, s_i, has an associated probability of failure, p_i. With these probabilities, we can easily assess the probability that a contamination incident will be detected by a particular sensor. Thus, it is straightforward to compute the expected impact of a contamination incident.

This formulation does not explicitly allow for the specification of probabilities of false detections. These probabilities do not impact the performance of a CWS during a contamination incident. Instead, they impact the day-to-day maintenance and use of the CWS; erroneous detections create work for the CWS users, which is an ongoing cost. The overall likelihood of false detections is simply a function of the sensors

that are selected. In cases where every sensor has the same likelihoods, this implies a simple constraint on the number of sensors.

4 Contamination Incidents and Impact Measures

This section describes how to simulate contamination incidents and compute contamination impacts, which are the first steps needed to setup and solve a sensor placement problem with SPOT. These two steps can be viewed as preprocessing or data preparation for sensor placement optimization. Thus, these steps can be performed prior to optimization, which is generally a more interactive, iterative process.

The following sections illustrate the capabilities of SPOT with the example in the C:\spot\examples\simple directory.

4.1 Simulating Contamination Incidents

To simulate contamination incidents, the **tevasim** (p. 53) command is utilized, which uses EPANET to perform an ensemble of contaminant transport simulations defined by a **TSG File** (p. 32). An ensemble of contamination scenarios for EPANET Example Net3 is defined in the file C:\spot\examples\simple\Net3.tsg. Contamination incidents are simulated for all network junctions, one for each hour of the day, and each contamination incident models an injection that continues for 24 hours. The `tevasim` command is run with the following command line:

```
tevasim --tsg Net3.tsg --tso Net3.tso Net3.inp Net3.out
```

This command generates three files: (a) Net3.tso, a binary TSO file that contains the contamination transport data, (b) Net3.sdx, a binary SDX file that provides an index into the TSO file, and (c) Net3.out, which provides a textual summary of the EPANET simulations and is the same as the report file (*.rpt) from EPANET.

4.2 Using tso2Impact

After running **tevasim** (p. 53) command, the output files, `Net3.tso` and `Net3.sdx`, can be used to compute one or more `IMPACT` files. An `IMPACT` file summarizes the consequence of each contamination incident in a manner that facilitates optimization. The **tso2Impact** (p. 54) command generates these files with the following command line:

```
tso2Impact --mc --vc --td --nfd --ec Net3 Net3.tso
```

This command generates IMPACT files for each of the five objectives specified: mass consumed (mc), volume consumed (vc), time to detection (td), number of failed detections (nfd) and extent of contamination (ec). For each IMPACT file (e.g. `Net3_mc.impact`), a corresponding `ID` file is generated to map the sensor placement ids back to the network junction labels (e.g. `Net3_mc.impact.id`).

The impact measures computed by `tso2Impact` represent the amount of impact that would occur up until the point where a contamination incident is detected. This computation assumes that sensors work perfectly (i.e., there are no false positive or false negative errors). However, we can generalize the sensor behavior in two ways. First, we can specify a detection threshold; contaminants are only detected above a specified concentration limit (the default limit is zero). Second, we can specify a response time, which accounts for the time needed to verify that a contamination has occurred and then inform the public (the default response time is zero). The contamination impact is computed at the time where the response has completed (the detection time plus response time), which is called the *effective response time*. For undetected incidents, the effective response time is simply the end of the contaminant transport simulation. The following illustrates how to specify these options:

```
tso2Impact --responseTime 60 --detectionLimit 0.1 --mc Net3 Net3.tso
```

This computes impacts for a 60 minute response time, with a 0.1 detection threshold. Note that the units for `-detectionLimit` are the same as for the MASS values that are specified in the TSG file.

Impacts from multiple TSO files can be combined to generate a single IMPACT file using the following syntax:

```
tso2Impact --detectionLimit 30000000 --detectionLimit 0.0001 --mc Net3 Net3_1a.tso Net3_1b.tso
```

Note that the value of 30000000 corresponds to the detection threshold for the contaminant described in `Net3_1a.tso` and 0.0001 is the detection threshold for the contaminant described in `Net3_1b.tso`. For example, this can be used to combine simulation results from different types of contaminants, in which the TSO files would have been generated from different TSG files. Murray et al. [12] use this technique to combine data from different tyes of contamination incidents into a single impact metric.

4.3 Impact Measures

After running **tevasim** (p. 53) command, the output files, `Net3.tso` and `Net3.sdx`, can be used to compute one or more `IMPACT` files. An `IMPACT` file summarizes the consequence of each contamination incident in a manner that facilitates optimization. A variety of objective measures are supported by `tso2Impact` to reflect the different criteria that decision makers could use in CWS design. For most of these criteria, there is a *detected* and *undetected* version of the objective. This difference concerns how undetected contamination incidents are modeled.

For example, the default time-to-detection objective, **td**, uses the time at which the EPANET simulations are terminated to define the time for incidents that are not detected. By contrast, the *detected* time-to-detection, **dtd**, simply ignores these incidents (they have impact zero). Sensor placement with the *detected* objective is somewhat more precise, but this objective needs to be optimized with a revised formulation that explicitly limits the fraction of incidents that are not detected by the sensors.

The following objectives are currently supported by `tso2Impact`:

- **ec** and **dec** - The extent of contaminated in the network. This is the total feet of pipes contaminated by the effective response time. An entire pipe is considered contaminated if contaminant enters the pipe at a given time step. For **ec**, the extent of contamination of an undetected incident is the extent of contamination at the point when the simulation terminates, while undetected contamination incidents are ignored for **dec**.

- **mc** and **dmc** - The mass of contaminant consumed by junctions in the network with nonzero demand. For **mc**, the mass of contaminant of an undetected incident is the mass of contaminant that has left the network via demand at the point when the simulation terminates, while undetected contamination incidents are ignored for **dmc**. This objective is typically measured in milligrams (the units used in the TSG file are mg/L). However, concentrations may also be interpreted; for example, we can treat this measure as a count of cells for a biological contaminant, where the TSG measurement is cells/L.

- **nfd** - The number of contamination incidents that are not detected by any sensor before the contaminant transport simulations terminate. NOTE: this measure is not affected by the response time option.

- **pe** and **dpe** - The number of individuals exposed to a contaminant. For **pe**, the population exposed for an undetected incident is the population exposed at the point when the simulation terminates, while undetected contamination incidents are ignored fo **dpe**.

- **pd** and **dpd** - The number of individuals that receive a dose of contaminant above a specified threshold. For **pd**, the population dosed by an undetected incident is the population dosed at the point when the simulation terminates, while for **dpd** the undetected contamination incidents are ignored.

- **pk** and **dpk** - The number of individuals killed by a contaminant. For **pk**, the population killed by an undetected incident is the population killed at the point when the simulation terminates, while for **dpk** the undetected contamination incidents are ignored.

- **td** and **dtd** - The time, in minutes, from the beginning of a contamination incident until the first sensor detects it. For **td**, the time-to-detection of an undetected incident is the time from the start of the incident until the end of the simulation, while undetected contamination incidents are ignored for **dtd**. NOTE: this measure is not affected by the response time option.

- **vc** and **dvc** - The volume of contaminated water consumed by junctions in the network with nonzero demand. For **vc**, the volume of contaminated water of an undetected incident is the volume of contaminated water consumed at the point when the simulation terminates, while undetected contamination incidents are ignored for **dvc**.

These health impact measures are computed with an auxiliary input file, **TAI**, that specifies parameters for a health impact model that predicts how a population is affected by exposure to a contaminant. The **TAI File** (p. 32) bio.tai specifies the nature of the contaminant and how it impacts human health. Further, this file specifies the fraction of the volume of water consumed at junctions that is consumed by humans. For example, consider the command line:

```
tso2Impact --pe Net3 Net3.tso bio.tai
```

4.4 Advanced Tools for Large Sensor Placements Problems

In some applications, the size of the IMPACT files is very large, which can lead to optimization models that cannot be solved on standard 32-bit workstations. SPOT includes several utilities that are not commonly used to address this challenge: the **scenarioAggr** (p. 45) executable aggregates similar contamination incidents, and the **filter_impacts** (p. 42) script filters out contamination incidents that have low impacts.

The **scenarioAggr** (p. 45) executable reads an IMPACT file, finds similar incidents, combines them, and writes out another IMPACT file. This aggregation technique combines two incidents that impact the same locations in the same order, allowing for the possibility that one incident continues to impact other locations. For example, two contamination incidents should travel in the same pattern if they differ only in the nature of the contaminant, though one may decay more quickly than the other. Aggregated incidents can be combined by simply averaging the impacts that they observe and updating the corresponding incident weight.

For example, consider the command:

```
scenarioAggr --numEvents=236 Net3_mc.impact
```

This creates the files `aggrNet3_mc.impact` and `aggrNet3_mc.impact.prob`; where the `Net3_mc.impact` file has 236 events. The file `aggrNet3_mc.impact` is the new IMPACT file, and the file `aggrNet3_mc.impact.prob` contains the probabilities of the aggregated incidents.

The **filter_impacts** (p. 42) script reads an impact file, filters out the low-impact incidents, rescales the impact values, and writes out another IMPACT file. The command:

```
filter_impacts --percent=5 Net3_mc.impact filtered.impact
```

generates an IMPACT file that contains the incidents whose impacts (without sensors) are the largest 5% of the incidents in `Net3_mc.impact`. Similarly, the `-num=k` option selects the k incidents with the largest impacts, and the option `-threshold=h` selects the incidents with the impacts greater than or equal to h.

The `filter_impacts` command also includes options to rescale the impact values. The `-rescale` option rescales impact values with a log-scale and the `-round` option rescales impact values to rounded log-scale values.

5 Sensor Placement Solvers

The SPOT sensor placement solvers are launched with the **sp** (p. 48) command. The `sp` command reads in one or more IMPACT files, and computes a sensor placement. Command-line options for `sp` can specify any of a set of performance or cost goals as the objective to be optimized, as well as constraints on performance and cost goals.

The `sp` command currently interfaces with three different sensor placement optimizers:

- **MIP solvers** - Several different MIP solvers can be used by the `sp` command: the commercial CPLEX solver and the open-source PICO solver. These optimizers use the MIP formulations to find globally optimal solutions. However, this may be a computationally expensive process (especially for large problems), and the size of the MIP formulation can become prohibitively large in some cases.

  Two different MIP solvers can be used: the public-domain PICO solver and the commercial PICO solver. PICO is included in distributions of SPOT.

- **GRASP Heuristic** - The GRASP heuristic performs sensor placement optimization without explicitly creating a MIP formulation. Thus, this solver uses much less memory, and it usually runs very quickly. Although the GRASP heuristic does not guarantee that a globally optimal solution is found, it has proven effective at finding optimal solutions to a variety of large-scale applications.

  Two different implementations of the GRASP solvers can be used: an ATT commercial solver (att_grasp) and an open-source implementation of this solver (snl_grasp).

- **GRASP Heuristic** - The GRASP heuristic performs sensor

- **Lagrangian Heuristic** - The Lagrangian heuristic uses the structure of the p-median MIP formulation (eSP) to find near-optimal solutions while computing a lower bound on the best possible solution.

The following sections provide examples that illustrate the use of the `sp` command. A complete description of `sp` is available in the Appendix. Note that this appendix includes a summary of the limitations of different solvers.

The `sp` command has many different options. The following examples show how different sensor placement optimization problems can be solved with `sp`. Note that these examples can be run in the `C:\spot\examples\simple` directory. The user needs to generate IMPACT files for these examples with the following commands:

```
tevasim --tsg Net3.tsg --tso Net3.tso Net3.inp Net3.out
tso2Impact --mc --vc --td --nfd --ec Net3 Net3.tso
```

5.1 A Simple Example

The following simple example illustrates the way that SPOT has been most commonly used. In this example, SPOT minimizes the extent of contamination (ec) while limiting the number of sensors (ns) to no more than 5. This problem formulation (eSP) can be efficiently solved with all solvers for modest-size distribution networks, and heuristics can effectively perform sensor placement on very large networks.

We begin by using the PICO solver to solve this problem, with the following command line:

```
sp --network=Net3 --objective=ec --ub=ns,5 --solver=pico
```

This specifies that network `Net3` is analyzed. The objective is to minimize `ec`, the extent of contamination, and an upper-bound of 5 is placed on `ns`, the number of sensors. The solver selected is `pico`, the PICO MIP solver.

This execution of the `sp` command uses the `Net3_ec.impact` file and creates the following files: `Net3.log`, a logfile for the optimization solver, and Net3.sensors, a file with the sensor placement locations. Also, `sp` generates the following output:

```
read_impact_files:  C:\spot\examples\simple\Net3_ec.impact

Number of Nodes                :  97
Number of Contamination Impacts: 9458

Running PICO...
PICO --debug=1 --lpType=clp   --tableInitFrac=0.05 --RRTrialsPerCall=8
--RRDepthThreshold=-1 --usingCuts=true  Net3.mod
C:\spot\examples\simple\Net3.dat
... PICO done
------------------------------------------------------------------------------
Sensor placement id:        22971
Number of sensors:          5
Total cost:                 0
Sensor node IDs:            19 28 54 63 75
Sensor junctions:           119 141 193 207 239

Impact File:                Net3_ec.impact
Number of events:           236
Min impact:                 0.0000
Mean impact:                8478.9674
Lower quartile impact:      0.0000
Median impact:              6949.0000
Upper quartile impact:      12530.0000
Value at Risk (VaR) (  5%): 25960.0000
TCE                 (  5%): 33323.2833
Max impact:                 42994.8000
------------------------------------------------------------------------------
------------------------------------------------------------------------------
Greedy ordering of sensors
------------------------------------------------------------------------------
54      29124.8004
19      18687.7292
63      11471.6750
75      9951.8699
28      8478.9674
Done with sp
```

The initial information up to the statment "... PICO done" is simply output about what solver is run and information from the solver output. The next information beginning with "Sensor placement id:" is generated by **evalsensor** (p. 40). This is a summary that describes the sensor placement and the performance of this sensor placement with respect to the impact measure that was minimized. This includes the following data:

- **Sensor placement id** - an integer ID used to distinguish this sensor placement
- **Number of sensors** - the number of sensors in the sensor placement
- **Total cost:** - the cost of the sensor placement, which may be nonzero if cost data is provided
- **Sensor node IDs** - the internal node indexes used by `sp`
- **Sensor junctions** - the EPANET junction labels for the sensor locations

The performance of the sensor placement is summarized for each IMPACT file used with `sp`. The impact statistics included are:

- **min** - The minimum impact over all contamination events. If we make the assumption that a sensor protects the node at which it is placed, then this measure will generally be zero.
- **mean** - The mean (or average) impact over all contamination events.
- **lower quartile** - 25% of contamination events, weighted by their likelihood, have an impact value less than this quartile.
- **median** - 50% of contamination events, weighted by their likelihood, have an impact value less than this quartile.
- **upper quartile** - 75% of contamination events, weighted by their likelihood, have an impact value less than this quartile.
- **VaR** - The value at risk (VaR) uses a user-defined percentile. Given $0.0 < \beta < 1.0$, VaR is the minimum value for which $100 * (1 - \beta)\%$ of contamination events have a smaller impact.
- **TCE** - The tailed-conditioned expectation (TCE) is the mean value of the impacts that are greater than or equal to VaR.
- **worst** - The value of the worst impact.

Finally, a greedy sensor placement is described by `evalsensor`, which takes the five sensor placements and places them one-at-a-time, minimizing the mean impact as each sensor is placed. This gives a sense of the relative priorities for these sensors.

The `evalsensor` command can evaluate a sensor placement for a wide variety of different objectives. For example, the command

```
evalsensor --nodemap=Net3.nodemap Net3.sensors Net3_ec.impact \
           Net3_mc.impact Net3_nfd.impact
```

will summarize the solution in the `Net3.sensors` file for the `ec`, `mc` and `nfd` impact measures.

The following example shows how to solve this same problem with the GRASP heuristic. This solver finds the same (optimal) solution as the MIP solver, though much more quickly.

```
sp --network=Net3 --objective=ec --ub=ns,5 --solver=snl_grasp

read_impact_files:  C:\spot\examples\simple\Net3_ec.impact
Note: witness aggregation disabled for grasp

Number of Nodes                :  97
Number of Contamination Impacts: 9458

Number of sensors=5
Objective=ec
Statistic=mean
Impact file=C:\spot\examples\simple\Net3_ec.impact
Delay=0
Running iterated descent heuristic for *perfect* sensor model
Iterated descent completed
--------------------------------------------------------------------------------
Sensor placement id:        23009
Number of sensors:          5
Total cost:                 0
Sensor node IDs:            19 28 54 63 75
Sensor junctions:           119 141 193 207 239

Impact File:                Net3_ec.impact
```

```
Number of events:           236
Min impact:                 0.0000
Mean impact:                8478.9674
Lower quartile impact:      0.0000
Median impact:              6949.0000
Upper quartile impact:      12530.0000
Value at Risk (VaR) (  5%): 25960.0000
TCE                 (  5%): 33323.2833
Max impact:                 42994.8000
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
Greedy ordering of sensors
--------------------------------------------------------------------------------
54      29124.8004
19      18687.7292
63      11471.6750
75      9951.8699
28      8478.9674
Done with sp
```

Finally, the following example shows how to solve this problem with the Lagrangian heuristic. This solver does not find as good a solution as the GRASP heuristic.

```
sp --network=Net3 --objective=ec --ub=ns,5 --solver=lagrangian


read_impact_files:  C:\spot\examples\simple\Net3_ec.impact

Number of Nodes                :  97
Number of Contamination Impacts: 9458

aggregateImpacts Net3.config 10000
Setting up Lagrangian data files...
Running UFL solver ...
ufl Net3_ec_agg.lag  6 0
--------------------------------------------------------------------------------
Sensor placement id:        27730
Number of sensors:          5
Total cost:                 0
Sensor node IDs:            15 17 19 21 66
Sensor junctions:           111 115 119 121 211

Impact File:                C:\spot\examples\simple\Net3_ec.impact
Number of events:           236
Min impact:                 0.0000
Mean impact:                12306.8229
Lower quartile impact:      0.0000
Median impact:              10410.0000
Upper quartile impact:      18100.0000
Value at Risk (VaR) (  5%): 41526.9000
TCE                 (  5%): 45057.9000
Max impact:                 49880.8000
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
Greedy ordering of sensors
--------------------------------------------------------------------------------
19      32174.3110
66      17854.3458
15      13583.0559
21      12929.5602
17      12306.8229
Done with sp
```

5.2 Computing a Bound on the Best Sensor Placement Value

The following example shows how a lower bound can be computed on the best possible sensor placement. That is, any sensor placement would have an expected impact greater than this value. A bound is computed with the following syntax:

```
sp --network=Net3 --objective=ec --ub=ns,5 --solver=pico --compute-bound

read_impact_files:  C:\spot\examples\simple\Net3_ec.impact

Number of Nodes                :  97
Number of Contamination Impacts: 9458

Running PICO...
PICO --debug=1 --lpType=clp   --onlyRootLP=true Net3.mod
C:\spot\examples\simple\Net3.dat
... PICO done
Computing a lower bound
Objective lower bound: 8478.96737288
Done with sp
```

5.3 Minimizing the Number of Sensors

We can "invert" the sensor placement problem by minimizing the number of sensors subject to a constraint on the extent of contamination. Note that the following example finds a solution with a single sensor that meets our goal of 40000 mean extent of contamination.

```
sp --network=Net3 --objective=ns --ub=ec,40000 --solver=pico

read_impact_files:  C:\spot\examples\simple\Net3_ec.impact

Number of Nodes                :  97
Number of Contamination Impacts: 9458

WARNING: Location aggregation does not work with side constraints
WARNING: Turning off location aggregation
Running PICO...
PICO --debug=1 --lpType=clp   --RRTrialsPerCall=8 --RRDepthThreshold=-1 --usingCuts=true --absTolerance=.99999  Net3.mod C:\spot
... PICO done
--------------------------------------------------------------------------------
Sensor placement id:        27738
Number of sensors:          1
Total cost:                 0
Sensor node IDs:            37
Sensor junctions:           161

Impact File:                C:\spot\examples\simple\Net3_ec.impact
Number of events:           236
Min impact:                 0.0000
Mean impact:                26901.9572
Lower quartile impact:      3940.0000
Median impact:              22450.0000
Upper quartile impact:      38855.0000
Value at Risk (VaR) (  5%): 71377.8000
TCE                 (  5%): 81046.0667
Max impact:                 103746.0000
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
Greedy ordering of sensors
--------------------------------------------------------------------------------
37      26901.9572
Done with sp
```

5.4 Fixing Sensor Placement Locations

Properties of the sensor locations can be specified with the `-sensor-locations` option. This options specifies a **Placement Locations File** (p. 36) that can control whether sensor locations are feasible or infeasible, and fixed or unfixed. For example, suppose the file `locations` contains

```
infeasible 193 119 141 207 239
fixed 161
```

The following example shows how these restrictions impact the solution. Compared to the first example above, we have a less-optimal solution, since we cannot use the sensor locations above and we are required to include junction `161`.

```
sp --network=Net3 --objective=ec --ub=ns,5 --solver=pico
        --sensor-locations=locations

read_impact_files:  C:\spot\examples\simple\Net3_ec.impact

Number of Nodes                :  97
Number of Contamination Impacts: 9458

Running PICO...
PICO --debug=1 --lpType=clp   --tableInitFrac=0.05 --RRTrialsPerCall=8
--RRDepthThreshold=-1 --usingCuts=true  Net3.mod
C:\spot\examples\simple\Net3.dat
... PICO done
--------------------------------------------------------------------------------
Sensor placement id:        22996
Number of sensors:          5
Total cost:                 0
Sensor node IDs:            17 33 37 50 66
Sensor junctions:           115 151 161 185 211

Impact File:                Net3_ec.impact
Number of events:           236
Min impact:                 0.0000
Mean impact:                9338.7119
Lower quartile impact:      0.0000
Median impact:              7640.0000
Upper quartile impact:      14120.0000
Value at Risk (VaR) (  5%): 27335.0000
TCE                 (  5%): 32282.3000
Max impact:                 45300.0000
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
Greedy ordering of sensors
--------------------------------------------------------------------------------
37      26901.9572
66      18192.6581
33      13958.3119
17      11281.2907
50      9338.7119
Done with sp
```

5.5 Robust Optimization of Sensor Locations

The following example demonstrates the optimization of sensor placements using the TCE measure. TCE is the mean value of the worst incidents in the ensemble being evaluated. Given a confidence level $1 - \beta$, TCE is the expectation of the worst impacts with probability β. Compared with our first example, we see that this finds a better solution in terms of TCE, although the mean performance is slightly worse.

```
sp --network=Net3 --objective=ec_tce --ub=ns,5 --solver=snl_grasp

read_impact_files:  C:\spot\examples\simple\Net3_ec.impact
Note: witness aggregation disabled for grasp

Number of Nodes                :  97
Number of Contamination Impacts: 9458

Number of sensors=5
Objective=ec
Statistic=tce
Impact file=C:\spot\examples\simple\Net3_ec.impact
Delay=0
Running iterated descent heuristic for *perfect* sensor model
Iterated descent completed
--------------------------------------------------------------------------------
Sensor placement id:        23005
Number of sensors:          5
Total cost:                 0
Sensor node IDs:            17 19 24 65 88
Sensor junctions:           115 119 127 209 267

Impact File:                Net3_ec.impact
Number of events:           236
Min impact:                 0.0000
Mean impact:                10266.1110
Lower quartile impact:      0.0000
Median impact:              10400.0000
Upper quartile impact:      16930.0000
Value at Risk (VaR) (  5%): 24199.0000
TCE                 (  5%): 26376.2167
Max impact:                 28564.8000
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
Greedy ordering of sensors
--------------------------------------------------------------------------------
88      30803.8140
19      19369.7636
65      12568.0822
17      11130.4161
24      10266.1110
Done with sp
```

Note that the greedy ordering of sensors is less useful in this case. Although we optimized to minimize TCE, the greedy ordering uses the mean value to select each sensor location.

5.6 Multi-Criteria Analysis

We now illustrate how `sp` supports multi-objective analysis through an iterative process. SPOT does not have a general "pareto search" optimizer. Instead, users can specify constraints with `sp` that ensure that previously optimized objectives are "close" to their previous values. In this way, the user can explicitly search for trade-offs between one-or-more performance objectives.

The examples above consider the extent-of-contamination objective. We can assess how well the sensor placements generated above minimize other objectives like the expected mass of contaminant consumed using `evalsensor`. Consider the solution generated by the previous example (which minimized `ec_tce`), which we have copied into the file `Net3_ec.sensors`.

```
evalsensor --nodemap=Net3.nodemap Net3_ec.sensors Net3_mc.impact
--------------------------------------------------------------------------------
Sensor placement id:        23112
Number of sensors:          5
Total cost:                 0
```

```
Sensor node IDs:            17 19 24 65 88
Sensor junctions:           115 119 127 209 267

Impact File:                Net3_mc.impact
Number of events:           236
Min impact:                 0.0000
Mean impact:                70895.2854
Lower quartile impact:      503.9170
Median impact:              83150.7000
Upper quartile impact:      144002.0000
Value at Risk (VaR) (  5%): 144271.0000
TCE                 (  5%): 144546.8333
Max impact:                 144693.0000
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
Greedy ordering of sensors
--------------------------------------------------------------------------------
65      71599.4274
88      71256.6780
24      71042.6323
17      70952.7213
19      70895.2854
```

The mean mass consumed is 70895, which is far from the optimal value of 21782 (which we computed separately). We revisit the robust optimization example in the previous section; we keep "extent of contamination - tce" as our primary objective, but we now impose a "side constraint" that precludes any solution that admits an average mass consumed of worse than 30,000 units. We do this as follows:

```
sp --network=Net3 --objective=ec_tce --ub=mc_mean,30000 --ub=ns,5 --solver=snl_grasp

read_impact_files:  C:\spot\examples\simple\Net3_ec.impact
read_impact_files:  C:\spot\examples\simple\Net3_mc.impact
Note: witness aggregation disabled for grasp

Number of Nodes                :  97
Number of Contamination Impacts: 9458

WARNING: Location aggregation does not work with side constraints
WARNING: Turning off location aggregation
Number of sensors=5
Objective=ec
Statistic=tce
Impact file=C:\spot\examples\simple\Net3_ec.impact
Delay=0
Running iterated descent heuristic for *perfect* sensor model
Iterated descent completed
--------------------------------------------------------------------------------
Sensor placement id:        23143
Number of sensors:          5
Total cost:                 0
Sensor node IDs:            4 15 29 68 81
Sensor junctions:           35 111 143 215 253

Impact File:                Net3_ec.impact
Number of events:           236
Min impact:                 0.0000
Mean impact:                14315.5322
Lower quartile impact:      1379.0000
Median impact:              10810.0000
Upper quartile impact:      21809.8000
Value at Risk (VaR) (  5%): 37915.8000
TCE                 (  5%): 48340.3667
Max impact:                 71329.0000

Impact File:                Net3_mc.impact
Number of events:           236
```

```
Min impact:                 0.0000
Mean impact:                29501.3226
Lower quartile impact:      139.0200
Median impact:              1766.1000
Upper quartile impact:      16476.3000
Value at Risk (VaR) (  5%): 144271.0000
TCE                 (  5%): 144276.3333
Max impact:                 144335.0000
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
Greedy ordering of sensors
--------------------------------------------------------------------------------
4       33437.5797
15      22324.5746
29      17100.4814
68      14912.0110
81      14315.5322
--------------------------------------------------------------------------------
Greedy ordering of sensors
--------------------------------------------------------------------------------
81      55207.4343
29      37684.3130
4       32849.0896
68      29738.0747
15      29501.3226
Done with sp
```

Note that the primary objective, minimizing the TCE of the "extent of contamination" measure, has gotten worse: it is now 48340 rather than 26376. However, our side constraint has been honored, and the mean mass consumed value is now 29501 rather than 70895.

5.7 Sensor Placements without Penalties

A fundamental issue for sensor placement is how to handle the fact that a limited budget of sensors will not be able to cover all possible incidents. SPOT addresses this issue by providing impact measures that integrate an impact 'penalty' for incidents that are not detected by a CWS design. Thus, in the previous examples there was an implicit trade-off between impact reduction and reduction in the number of contamination incidents that are detected.

SPOT also includes impact measures that do not contain these penalties, which allows a user to more directly assess the performance of a CWS design in the context where detections have occured. For example, the time-to-detection measure (td) includes a penalty for undetected incidents, but the detected-time-to-detection measure (dtd) has no penalty (or, more precisely, a zero penalty).

For example, consider the simple example above, which minimizes the extent of contamination. We apply `evalsensors` to the final solution to evaluate the **ec**, **dec** and **nfd** impact measures:

```
evalsensor --nodemap=Net3.nodemap Net3_orig.sensors Net3_ec.impact
Net3_dec.impact Net3_nfd.impact

--------------------------------------------------------------------------------
Sensor placement id:        3789
Number of sensors:          5
Total cost:                 0
Sensor node IDs:            19 28 54 63 75
Sensor junctions:           119 141 193 207 239

Impact File:                Net3_ec.impact
Number of events:           236
Min impact:                 0.0000
Mean impact:                8478.9674
Lower quartile impact:      0.0000
```

```
Median impact:              6949.0000
Upper quartile impact:      12530.0000
Value at Risk (VaR) (  5%): 25960.0000
TCE                 (  5%): 33323.2833
Max impact:                 42994.8000

Impact File:                Net3_dec.impact
Number of events:           236
Min impact:                 0.0000
Mean impact:                8184.5182
Lower quartile impact:      0.0000
Median impact:              6949.0000
Upper quartile impact:      12530.0000
Value at Risk (VaR) (  5%): 25960.0000
TCE                 (  5%): 33323.2833
Max impact:                 42994.8000

Impact File:                Net3_nfd.impact
Number of events:           236
Min impact:                 0.0000
Mean impact:                0.2500
Lower quartile impact:      0.0000
Median impact:              0.0000
Upper quartile impact:      0.0000
Value at Risk (VaR) (  5%): 1.0000
TCE                 (  5%): 1.0000
Max impact:                 1.0000
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
Greedy ordering of sensors: Net3_ec.impact
--------------------------------------------------------------------------------
54      29124.8004
19      18687.7292
63      11471.6750
75      9951.8699
28      8478.9674
--------------------------------------------------------------------------------
Greedy ordering of sensors: Net3_dec.impact
--------------------------------------------------------------------------------
19      4845.7771
28      3613.4678
54      10489.1614
63      7097.6114
75      8184.5182
--------------------------------------------------------------------------------
Greedy ordering of sensors: Net3_nfd.impact
--------------------------------------------------------------------------------
75      0.2712
28      0.2500
19      0.2500
54      0.2500
63      0.2500
```

In this example, the final sensor placement fails to detect 25% of the incidents. It is noteworthy that this does not impact the mean performance very much, since the impact penalty has led to a final solution that fails to detect few incidents with high penalties.

Note that minimizing **dtd** does not really make sense. With zero sensors, you detect no incidents, which means that the final impact measurement is zero! Thus, minimizing **dtd** requires the additional constraint on the number of failed detections (nfd) as well as a limit on the number of sensors (or total sensor costs).

Only the 'pico' SPOT solver currently supports optimization with 'detected' impact measures. For example:

```
sp --network=Net3 --objective=dec --ub=ns,5 --ub=nfd,0.25 --solver=pico
```

```
Number of Nodes                :  97
Number of Contamination Impacts: 9458

WARNING: Location aggregation does not work with side constraints
WARNING: Turning off location aggregation
Running PICO...
PICO --debug=1 --lpType=clp   --RRTrialsPerCall=8
--RRDepthThreshold=-1 --feasibilityPump=false --usingCuts=true
Net3.mod C:\spot\examples\simple\Net3.dat
... PICO done
--------------------------------------------------------------------------------
Sensor placement id:        23466
Number of sensors:          5
Total cost:                 0
Sensor node IDs:            19 28 54 63 75
Sensor junctions:           119 141 193 207 239

Impact File:                Net3_dec.impact
Number of events:           236
Min impact:                 0.0000
Mean impact:                8184.5182
Lower quartile impact:      0.0000
Median impact:              6949.0000
Upper quartile impact:      12530.0000
Value at Risk (VaR) (  5%): 25960.0000
TCE                 (  5%): 33323.2833
Max impact:                 42994.8000

Impact File:                Net3_nfd.impact
Number of events:           236
Min impact:                 0.0000
Mean impact:                0.2500
Lower quartile impact:      0.0000
Median impact:              0.0000
Upper quartile impact:      0.0000
Value at Risk (VaR) (  5%): 1.0000
TCE                 (  5%): 1.0000
Max impact:                 1.0000
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
Greedy ordering of sensors: Net3_dec.impact
--------------------------------------------------------------------------------
19      4845.7771
28      3613.4678
54      10489.1614
63      7097.6114
75      8184.5182
--------------------------------------------------------------------------------
Greedy ordering of sensors: Net3_nfd.impact
--------------------------------------------------------------------------------
75      0.2712
28      0.2500
19      0.2500
54      0.2500
63      0.2500
Done with sp
```

5.8 Limited-Memory Sensor Placement Techniques

Controlling memory usage is a critical issue for solving large sensor placement formulations. This is a particular challenge for MIP methods, but both the GRASP and Lagrangian heuristics can have difficultly solving very large problems on standard workstations. A variety of mechanisms have been integrated into `sp` to reduce the problem representation size while preserving the structure of the sensor placement problem.

The **scenarioAggr** (p. 45) method described in the previous section is one possible strategy. This tool

compresses the impact file while preserving the fundamental structure of the impact file and it is appropriate when optimizing for mean performance objectives. Similarly, the **filter_impacts** (p. 42) script can limit the sensor placement to only consider contamination incidents that are "sufficiently bad" in the worst-case. Another strategy is to limit the number of sensor placements, using the `-sensor-locations` option described above, since eliminating feasible locations reduces the problem representation used by the `sp` solvers.

Two other strategies are also supported by `sp`. First, the GRASP heuristic has several options for controlling how memory is managed. The `-grasp-representation` option can be used to control how the local search steps are performed. By default, a dense matrix is precomputed to perform local search steps quickly, but a sparse matrix can be used to perform local search with less memory. Also, the GRASP heuristic can be configured to use the local disk to store this matrix. It should be noted that the Lagrangian heuristic requires less memory than the GRASP heuristic, and thus similar techniques have not been developed for it.

Second, the *witness aggregation* technique can be used to limit the size of the sensor placement formulation. This term refers to the variables in the MIP formulation that "witness" a contamination event. By default, variables that witness contamination events with the same impact are aggregated, and this typically reduces the MIP constraint matrix by a significant amount. Further reductions can be performed with more aggressive aggregations.

To illustrate the use of witness aggregation, we generated impact files with the `C:\spot\etc\tsg\hourly.tsg` TSG file. The following table illustrates the use of the two witness aggregation options when optimizing the mean extent of contamination: `-aggregation-percent` and `-aggregation-ratio` (used with the `-distinguish-detection` option, which helps this aggregation option). The second line of data in this table is the default aggregation, which has about half as many non-zero values in the MIP constraint matrix. Both the percent and ratio aggregation strategies effectively reduce the problem size while finding near-optimal solutions.

Aggr Type	Aggr Value	Binary Vars	MIP Nonzeros	Solution Value
None	NA	97	220736	8525
Percent	0.0	97	119607	8525
Percent	0.125	97	49576	9513
Ratio	0.125	97	12437	10991

5.9 Evaluating a Sensor Placement

The **evalsensor** (p. 40) executable takes sensor placements in a **Sensor Placement File** (p. 33) and evaluates them using data from an **Impact File** (p. 33) (or a list of impact files). This executable measures the performance of each sensor placement with respect to the set of possible contamination locations. This analysis assumes that probabilities have been assigned to these contamination locations, and if no probabilities are given then uniform probabilies are used by `evalsensor`.

The following example illustrates the use of `evalsensor` after running the first sensor placement optimization example.

```
evalsensor --nodemap=Net3.nodemap Net3_orig.sensors Net3_ec.impact
Net3_mc.impact
------------------------------------------------------------------------------
Sensor placement id:        5511
Number of sensors:          5
Total cost:                 0
Sensor node IDs:            19 28 54 63 75
Sensor junctions:           119 141 193 207 239

Impact File:                Net3_ec.impact
Number of events:           236
Min impact:                 0.0000
```

```
Mean impact:                8478.9674
Lower quartile impact:      0.0000
Median impact:              6949.0000
Upper quartile impact:      12530.0000
Value at Risk (VaR) (  5%): 25960.0000
TCE                 (  5%): 33323.2833
Max impact:                 42994.8000

Impact File:                Net3_mc.impact
Number of events:           236
Min impact:                 0.0000
Mean impact:                43636.7076
Lower quartile impact:      220.0020
Median impact:              1909.9500
Upper quartile impact:      105031.0000
Value at Risk (VaR) (  5%): 144271.0000
TCE                 (  5%): 144345.0000
Max impact:                 144477.0000
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
Greedy ordering of sensors
--------------------------------------------------------------------------------
54      29124.8004
19      18687.7292
63      11471.6750
75      9951.8699
28      8478.9674
--------------------------------------------------------------------------------
Greedy ordering of sensors
--------------------------------------------------------------------------------
75      59403.2616
28      44478.4678
63      43854.6979
54      43659.7307
19      43636.7076
```

The `evalsensors` command can also evaluate a sensor placement in the case where sensors can fail, and there is some small number of different classes of sensors (grouped by false negative probability). Consider the `Net3.imperfectsc` file, which defines different categories of sensor failures:

```
1 0.25
2 0.50
3 0.75
4 1.0
```

Sensors of class "1" give false negative readings 25% of the time, sensors of class "2" give them 50% of the time, etc.

Once failure classes have been defined, the junctions of the network are assigned to classes. This is done with another file (a "junction class" or jc file), like `Net3.imperfectjc`.

```
1 1
2 1
3 1
4 1
5 1
6 1
7 1
8 1
9 1
10 1
11 1
12 1
```

```
13 1
14 1
15 1
16 1
17 1
18 1
19 1
20 1
....
....
```

Given the junction classes, we can run `evalsensor` to determine the expected impact of a sensor placement, given that sensors may fail. Again, using the solution from the original example:

```
evalsensor --nodemap=Net3.nodemap --sc-probabilities=Net3.imperfectsc
--scs=Net3.imperfectjc Net3_orig.sensors Net3_ec.impact

------------------------------------------------------------------------------
Sensor placement id:        5511
Number of sensors:          5
Total cost:                 0
Sensor node IDs:            19 28 54 63 75
Sensor junctions:           119 141 193 207 239

Impact File:                Net3_ec.impact
Number of events:           236
Min impact:                 0.0000
Mean impact:                17161.8656
Lower quartile impact:      3940.0000
Median impact:              15307.2500
Upper quartile impact:      26537.2156
Value at Risk (VaR) (  5%): 47637.7773
TCE                 (  5%): 54977.0644
Max impact:                 75509.9043
------------------------------------------------------------------------------
------------------------------------------------------------------------------
Greedy ordering of sensors
------------------------------------------------------------------------------
54      36553.6321
63      25655.0376
28      21209.3084
75      18810.1629
19      17161.8656
```

Note that the mean impact of this "extent of contamination" changes dramatically if sensors are allowed to fail. The original solution, 8478 pipe feet, was misleading if sensors fail according to the probabilities we have assigned. With sensor failures, the expected impact is 17161 pipe feet – more than twice the "perfect sensor" impact.

5.10 Sensor Placement with Imperfect Sensors

The GRASP heuristics in SPOT can optimize sensor placements that take into account sensor failures. For example, we can perform sensor placement optimization with imperfect sensors using the `Net3.imperfectsc` and `Net3.imperfectjc` files defined in the previous section.

```
sp --network=Net3 --objective=ec --ub=ns,5 --imperfect-scfile=Net3.imperfectsc
--imperfect-jcfile=Net3.imperfectjc --solver=snl_grasp

read_impact_files:  C:\spot\examples\simple\Net3_ec.impact
Note: witness aggregation disabled for grasp
```

```
Number of Nodes                :  97
Number of Contamination Impacts: 9458

Number of sensors=5
Objective=ec
Statistic=mean
Impact file=C:\spot\examples\simple\Net3_ec.impact
Delay=0
Running iterated descent heuristic for *imperfect* sensor model
Iterated descent completed
-------------------------------------------------------------------------------
Sensor placement id:        9285
Number of sensors:          5
Total cost:                 0
Sensor node IDs:            33 63 75 83 87
Sensor junctions:           151 207 239 257 265

Impact File:                Net3_ec.impact
Number of events:           236
Min impact:                 0.0000
Mean impact:                13414.2479
Lower quartile impact:      3610.0000
Median impact:              11690.0000
Upper quartile impact:      20096.0000
Value at Risk (VaR) (  5%): 36467.5500
TCE                 (  5%): 44420.1990
Max impact:                 63324.0500
-------------------------------------------------------------------------------
-------------------------------------------------------------------------------
Greedy ordering of sensors
-------------------------------------------------------------------------------
87      27984.5508
63      22080.0939
83      16853.0790
75      14955.9915
33      13414.2479
Done with sp
```

After this optimization, the mean impact is 13414 pipe feet rather than the 17161 pipe feet value for the solution optimized with perfect sensors. Thus, it is clear that the GRASP heuristic makes different choices if the sensors are imperfect.

5.11 Summary of Solver Features

The following table highlights the capabilities of the SPOT optimizers. The previous examples illustrate SPOT's capabilities, but the advanced features in SPOT are not available for all optimizers.

Solver Feature	**MIP**	**GRASP**	**Lagrangian**
Minimize mean impact	YES	YES	YES
Minimize worst impact	YES	YES	NO
Minimized number of sensors	YES	NO	NO
Robust objectives	YES	YES	NO
Side-constraints	YES	YES	YES
Fixed/Invalid locations	YES	YES	YES
Witness aggregation	YES	NO	YES
Incident probabilities	YES	NO	YES
Incident aggregation	YES	NO	YES
Sparse data management	NO	YES	NO
Imperfect sensor model	NO	YES	NO
Computes lower bound	YES	NO	YES

6 File Formats

6.1 TSO File

- **Desription:** provides a compact representation of simulation results.
- **Format:** binary
- **Created By:** `tevasim`
- **Used By:** `tso2Impact`
- **Details:** The format of TSO files is described in C:\spot\doc\TEVAUtil.doc.

6.2 SDX File

- **Desription:** provides an index into a TSO file.
- **Format:** binary
- **Created By:** `tevasim`
- **Used By:** `tso2Impact`
- **Details:** SDX files provide an index file that contains information about at what byte offset in which TSO file a particular injection scenario's results are located. The format of SDX files is described in C:\spot\doc\Threat\ Simulator.doc.

6.3 TSG File

- **Desription:** specifies how an ensemble of EPANET simulations will be performed.
- **Format:** ascii
- **Created By:** SPOT user
- **Used By:** `tevasim`
- **Details:** Each line of a TSG file specifies injection locations, the injection mass, and the injection time-frame:

  ```
  NZD MASS <injection-mass> <start-time> <end-time>
  ```

  This format is described in detail in C:\spot\doc\Threat\ Simulator.doc.

6.4 TAI File

- **Desription:** describes the information needed for assessing health impacts
- **Format:** ascii
- **Created By:** SPOT user

- **Used By:** `tevasim`
- **Details:** A TAI provides information needed for assessing health impacts. This file is only required for impact values like `pe` that involve health impacts. The format of TAI files will be described in C:\spot\doc\threatAssess_readme.txt when health impacts are integrated into the TEVA-SPOT Toolkit release.

6.5 Sensor Placement File

- **Desription:** describes one or more sensor placements
- **Format:** ascii
- **Created By:** `sp`
- **Used By:** `evalsensor`
- **Details:**

  Lines in a sensor placement file that begin with the '#' character are assumed to be comments. Otherwise, lines of this file have the format

  ```
  <sp-id> <number-of-sensors> <node-index-1> ...
  ```

  The sensor placement ID is used to identify sensor placements in the file. Sensor placements may have differing numbers of sensors, so each line contains this information. The node indices map to values in the **Node File** (p. 34).

6.6 Impact File

- **Desription:** describes the impact of a contamination event at the point that it is witnessed through a water distribution network.
- **Format:** ascii
- **Created By:** `tso2Impact`
- **Used By:** `sp` and `evalsensor`
- **Details:** An IMPACT file describes the impact of a contamination event at the point that it is witnessed throughout a water distribution network. Specifically, the witness events are assumed to occur at junctions in the network.

  The first line of an IMPACT file contains the number of events. The next line specifies the types of delayed impacts provided in this file, with the format:

  ```
  <number-of-delays> <delay-time1> ... <delay-timeN>
  ```

  The delay times are in minutes. (Currently, the SPOT utilities only support a single delay time.)

  Subsequent lines have the format

  ```
  <scenario-index> <node-index> <time-of-detection> <impact-value>
  ```

  The node index is the index of a witness location for the attack. A scenario ID maps to a line in the network **Scenario File** (p. 34). A node index maps to a line in the network **Node File** (p. 34). The time of detection is in minutes. The value of impacts are in the corresponding units for each impact measure. The different impact measures in each line correspond to the different delays that have been computed.

6.7 LAG File

- **Desription:** A sparse matrix file used by the UFL Lagrangian solver
- **Format:** ascii
- **Created By:** `setupIPData`
- **Used By:** `lagrangian`
- **Details:** This is a variant of the IMPACT format. Conceptually, it can be viewed as a transpose of the matrix specified in an IMPACT file. The first line specifies the number of locations, the number of events, and the impact values:

  ```
  <num-locations> <num-events> <impact>
  ```

  These impact values differ from the values in the IMPACT file, in that they are normalized by the probability of the event. Subsequent lines describe the impact of each event:

  ```
  <location> <event> <impact>
  ```

  Note that the location and event indices are indexed starting at 1.

6.8 Scenario File

- **Desription:** The scenario file provides auxillary information about each contamination incident.
- **Format:** ascii
- **Created By:** `tso2Impact`
- **Used By:** `evalsensor`
- **Details:** The scenario file provides auxillary information about each contamination scenario. Each line of this file has the format:

  ```
  <node-index> <EPANET-ID> <source-type> <source-start-time> <source-stop-time> <source-strength>
  ```

  The node index maps to the network **Node File** (p. 34), and the EPANET ID provides this information (redundantly). The scenario start and stop are in minutes, and these values are relative to the start of the EPANET simulation. The source type is the injection mode for an attack, e.g., flow-paced or fixed-concentration. The source strength is the concentration of contaminant at the attack source.

6.9 Node File

- **Desription:** provides a mapping from the indices used for sensor placement to the junction IDs used within EPANET
- **Format:** ascii
- **Created By:** `tso2Impact`
- **Used By:** `evalsensor` and sensor placement solvers
- **Details:** The node file provides a mapping from the indices used for sensor placement to the IDs used within EPANET. Each line of this file has the format:

```
<node-index> <EPANET-ID>
```

This mapping is generated by **tso2Impact** (p. 54), and all sensor placement solvers subsequently use the node indices internally.

6.10 Sensor Placement Configuration File

- **Desription:** a configuration file used to define a sensor placement problem
- **Format:** ascii
- **Created By:** sp
- **Used By:** setupIPData
- **Details:** The sensor placement configuration file is generated by the **sp** (p. 48) solver interface, and it contains all of the information that is needed to define a sensor placement problem. The file has the following format:

```
<number-of-junctions> <response-delay>
<number-of-goals>
<goal-name> <goal-filename> <compression-threshold> <compression-percentage> <attack-collapse-flag>\
        <number-of-measures>\
<measure-name> ... <measure-name> <objective-flag>\
<bound-value> ... <bound-value>
<fixed-sensor-placements>
<invalid-sensor-placements>
<cost-values>
```

The values in this file correspond to the command-line arguments of the **sp** (p. 48) solver. Compression threshold or percentage refers to node aggregation values. The attack-collapse-flag is a 0 or 1 value in the configuration file, indicating whether compression/aggregation can make an attack trivial (single supernode equivalent to no detection). The <fixed-sensor-placements>, <invalid-sensor-placements> and <cost-values> data sets are simply an import of the data from the corresponding files that are specified within the **sp** (p. 48) solver interface.

6.11 Sensor Placement Costs File

- **Desription:** specifies the costs for installing sensors at different junctions throughout a network
- **Format:** ascii
- **Created By:** SPOT user
- **Used By:** sp
- **Details:** Each line of this file has the format:

```
<EPANET-ID> <cost>
```

Junctions not explicitly enumerated in this file are assumed to have zero cost unless the ID '__-default__' is specified. For example:

```
__default 1.0
```

This example would specify that all un-enumerated junctions have a cost of 1.0.

6.12 Placement Locations File

- **Desription:** specifies whether sensor placements are fixed and whether locations are feasible sensor placement
- **Format:** ascii
- **Created By:** SPOT user
- **Used By:** `sp`
- **Details:** Each line of this file has the format:

  ```
  <keyword> <EPANET-ID> ... <EPANET-ID>
  ```

  The valid keywords are **feasible**, **infeasible**, **fixed** and **unfixed**. These keywords correspond to two semantic states for each location: (1) the feasibility of sensor placement and (2) whether a sensor placement is forced. The semantics of these keywords are as follows:

  - **feasible** - the specified locations are **feasible** and **unfixed**
  - **infeasible** - the specified locations are **infeasible** and **unfixed**
  - **fixed** - the specified locations are constrained to contain a sensor (**fixed** and **feasible**)
  - **unfixed** - the specified locations are not constrained to contain a sensor (**unfixed** and **feasible**)

  The locations are EPANET-IDs from the network model. Additionally, the keyword **ALL** or $*$ can be used to specify that all network locations are to be processed.

  A location file is processed sequentially. The initial state is that all locations are feasible and unfixed. Subsequently, each line updates the state of the locations, given the state defined by the previous lines of this file. For example, the file:

  ```
  infeasible ALL
  feasible A B C
  ```

  makes all locations infeasible except for locations A, B and C. Similarly

  ```
  fixed ALL
  feasible A B C
  ```

  makes all locations fixed except for locations A, B and C; the **feasible** keyword has the same semantics as the **unfixed** keyword.

6.13 Sensor Class File

- **Desription:** contains false-negative probabilities for different types of sensors
- **Format:** ascii
- **Created By:** SPOT user
- **Used By:** `sp`
- **Details:** The file has format:

  ```
  <class id> <false negative probability>
  <class id> <false negative probability>
  ....
  ```

For example, the following file defines a failure class 1, with a false negative rate of 25 percent, and a failure class 2 with a false negative rate of 50 percent:

```
1 0.25
2 0.5
....
```

6.14 Junctions Class File

- **Desription:** provides the mapping from EPANET junction IDs to failure classes
- **Format:** ascii
- **Created By:** EPANET user
- **Used By:** `sp`
- **Details:** When a sensor class file is being used, the "junction class" file provides the mapping from junction (node) id's to failure classes. The format of this file is:

  ```
  <node id> <failure class>
  <node id> <failure class>
  ....
  ```

For example, supposing that junction 1 is of class 2, junction 2 is of class 1, and junction 3 is of class 1:

```
1 1
2 2
3 1
....
```

Bibliography

[1] R. Bahadur, W. B. Samuels, W. Grayman, D. Amstutz, and J. Pickus. Pipelinenet: A model for monitoring introduced contaminants in a distribution system. In *Proc. World Water and Environmental Resources Congress*. ASCE, 2003.

[2] J. Berry, R. Carr, W. Hart, V. Leung, C. Phillips, and J. Watson. On the placement of imperfect sensors in municipal water networks. In *Proceedings of the 8th Symposium on Water Distribution Systems Analysis*. ASCE, 2006.

[3] J. Berry, L. Fleischer, W. E. Hart, C. A. Phillips, and J.-P. Watson. Sensor placement in municipal water networks. *J. Water Resources Planning and Management*, 131(3):237–243, 2005.

[4] J. Berry, W. E. Hart, C. A. Phillips, J. Uber, and T. Walski. Water quality sensor placement in water networks with budget constraints. In *Proc. World Water and Environment Resources Conference*, 2005.

[5] J. Berry, W. E. Hart, C. E. Phillips, J. G. . Uber, and J.-P. Watson. Sensor placement in municiple water networks with temporal integer prog ramming models. *J. Water Resources Planning and Management*, 132(4):218–224, 2006.

[6] J. W. Berry, W. E. Hart, and C. A. Phillips. Scalability of integer programming computations for sensor placement in municipal water networks. In *Proc. World Water and Environmental Resources Congress*. ASCE, 2005.

[7] R. Carr, H. Greenberg, W. Hart, G. Konjevod, E. Lauer, H. Lin, T. Morrison, and C. Phillips. Robust optimization of contaminant sensor placement for community water systems. *Mathematical Programming*, 107(1):337–356, 2006.

[8] J. R. Chastain, Jr. *A Heuristic Methodology for Locating Monitoring Stations to Detect Contamination Events in Potable Water Distribution Systems*. PhD thesis, University of South Florida, 2004.

[9] W. E. Hart, J. Berry, R. Murray, C. A. Phillips, L. A. Riesen, and J.-P. Watson. SPOT: A sensor placement optimization toolkit for drinking water contaminant warning system design. Technical Report SAND2007-4393, Sandia National Laboratories, 2007.

[10] P. Mirchandani and R. Francis, editors. *Discrete Location Theory*. John Wiley and Sons, 1990.

[11] K. Morley, R. Janke, R. Murray, and K. Fox. Drinking water contamination warning systems: Water utilities driving water research. *J. AWWA*, pages 40–46, 2007.

[12] R. Murray, J. Berry, and W. E. Hart. Sensor network design for contamination warning systems: Tools and applications. In *Proc. AWWA Water Security Congress*, 2006.

[13] R. Murray, J. Uber, and R. Janke. Modeling acute health impacts resulting from ingestion of contaminated drinking water. *J. Water Resources Planning and Management: Special Issue on Drinking Water Distribution Systems Security*, 2006.

[14] A. Ostfeld and E. Salomons. Optimal layout of early warning detection stations for water distribution systems security. *J. Water Resources Planning and Management*, 130(5):377–385, 2004.

[15] A. Ostfeld, J. G. Uber, E. Salomons, J. W. Berry, W. E. Hart, C. A. Phillips, J.-P. Watson, G. Dorini, P. Jonkergouw, Z. Kapelan, F. di Pierro, S.-T. Khu, D. Savic, D. Eliades, M. Polycarpou, S. R. Ghimire, B. D. Barkdoll, R. Gueli, J. J. Huang, E. A. McBean, W. James, A. Krause, J. Leskovec, S. Isovitsch, J. Xu, C. Guestrin, J. VanBriesen, M. Small, P. Fischbeck, A. Preis, M. Propato, O. Piller, G. B. Trachtman, Z. Y. Wu, and T. Walski. The battle of the water sensor networks (BWSN): A design challenge for engineers and algorithms. *J Water Resources Planning and Management*, 2007. (submitted).

[16] R. T. Rockafellar and S. Uryasev. Conditional value-at-risk for general loss distributions. *J. of Banking and Finance*, 26(7):1443–1471, 2002.

[17] L. A. Rossman. The EPANET programmer's toolkit for analysis of water distribution systems. In *Proceedings of the Annual Water Resources Planning and Management Conference*, 1999. Available at `http://www.epanet.gov/ORD/NRMRL/wswrd/epanet.html`.

[18] N. Topaloglou, H. Vladimirou, and S. Zenios. CVaR models with selective hedging for international asset allocation. *J. of Banking and Finance*, (26):1535–1561, 2002.

[19] G. B. Trachtman. A "strawman" common sense approach for water quality sensor site selection. In *Proc. 8th Annual Water Distribution System Analysis Symposium*, 2006.

[20] USEPA. WaterSentinel System Architecture. Technical report, U.S. Environmental Protection Agency, 2005.

[21] J.-P. Watson, H. J. Greenberg, and W. E. Hart. A multiple-objective analysis of sensor placement optimization in water networks. In *Proc. World Water and Environment Resources Conference*, 2004.

[22] J.-P. Watson, W. E. Hart, and R. Murray. Formulation and optimization of robust sensor placement problems for contaminant warning systems. In *Proc. Water Distribution System Symposium*, 2006.

A Executable evalsensor

A.1 Overview

The **evalsensor** executable is used to compute information about the impact of contamination events for one (or more) sensor placements.

A.2 Command-Line Help

```
Usage: evalsensor [options...] <sensor-file> <impact-file1>
                              [<impact-file2>...]
Usage: evalsensor [options...] none <impact-file1> [<impact-file2>...]
A command to read in one or more sensor placements and summarize their
performance according to various metrics.

options:

      --all-locs-feasible     A boolean flag that indicates that all locations
                              are treated as feasible.
      --costs                 A file with the cost information for each
                              location id.
      --debug                 A boolean flag that adds output information
                              about each event.
      --event-probabilities   A file with the probability of the different
                              contamination scenarios.
      --format                The type of output that the evaluation will
                              generate:
                               cout  Generates output that is easily read.
                                     (default)
                               xls   Generates output that is easily
                                     imported into a MS Excel spreadsheet.
                               xml   Generates an XML-formated output that
                                     is used to communicate with the TEVA
                                     GUI.  (Not currently supported.)
      --gamma                 The fraction of the tail distribution used to
                              compute the VaR and TCE performance measures.
                              (The default value is 0.05).
  -h, --help                  Display usage information
      --nodemap               A file with the node map information, for
                              translating node IDs into junction labels.
  -r, --responseTime          This parameter indicates the number of minutes
                              that are needed to respond to the detection of a
                              continant.  As the response time increases, the
                              impact increases because the contaminant affects
                              the network for a greater length of time. Unit:
                              minutes.
      --sc-probabilities      A file with the probability of detection for
                              each sensor category.
      --scs                   A file with the sensor category information for
                              each location id.
      --version               Display version information

arguments:

  sensor-file:  A sensor placement file, which contains one or more sensor
      placements that will be evaluated.  If 'none' is specified, then
      evalsensor will evaluate impacts with no sensors.

  impact-file:  A file that contains the impact data concerning a
      contamination event.  If one or more impact files are specified, then
      evaluations are performed for each impact separately.
```

```
Note that options like 'responseTime' can be specified with the syntax
'--responseTime 10.0' or '--responseTime=10.0'.
```

A.3 Description

The **evalsensor** executable takes sensor placements in a **Sensor Placement File** (p. 33) and evaluates them using data from an **Impact File** (p. 33) (or a list of impact files). This executable measures the performance of each sensor placement with respect to the set of possible contamination locations.

See Section **here** (p. 27) for further description of this command.

A.4 Notes

None.

B Executable filter_impacts

B.1 Overview

The **filter_impacts** script filters out the low-impact incidents from an impact file.

B.2 Usage

```
filter_impacts [options...] <impact-file> <out-file>
```

B.3 Options

```
 --threshold=<val>
Keep only the incidents whose undetected impact is above a
specified threshold.

 --percent=<num>
Keep the <num> percent of the incidents with the worst
undetected impact.

 --num=<num>
Keep the <num> incidents with the worst
undetected impact.

 --rescale
Rescale the impacts using a log10 scale.

 --round
Round input values to the nearest integer.
```

B.4 Arguments

```
 <impact-file>
The input impact file.

 <out-file>
The output impact file.
```

B.5 Description

The **filter_impacts** command reads an impact file, filters out the low-impact incidents, rescales the impact values, and writes out another impact file.

B.6 Notes

None.

C Executable PICO

C.1 Overview

The **PICO** executable used by `sp` that solves linear programs and mixed-integer linear programs.

C.2 Usage

```
PICO [options...] <input-file>
```

C.3 Options

Documentation of **PICO** options is available from the PICO User Manual, which is available from http://software.sandia.gov/Acro/PICO.

C.4 Description

PICO is a general-purpose solver for linear and integer programs. This command is not directly used by the user.

PICO uses public-domain software components, and thus it can be used without licensing restrictions. The integer programming format used in SPOT is defined with the AMPL modeling language. PICO integrates the GLPK mathprog problem reader, which is compatible with a subset of the AMPL modeling language. This enables PICO to process an integer programming formulation in SPOT that can also be used with AMPL.

C.5 Notes

- On large-scale tests, we have noted that PICO's performance is often limited by the performance of the public-domain LP solvers that it employs. In some cases, we have noted that these solvers can be over 100 times slower than the state-of-the-art CPLEX LP solver.

D Executable randomsample

D.1 Overview

The **randomsample** executable heuristically solves p-median formulations of the sensor placement problem.

D.2 Usage

```
   randomsample <sp-configuration-file> <num-sample> <random-seed>
<impact-file-representation> <time-limit> [<solution-file>]
```

D.3 Arguments

```
   <sp-configuration-file>
   The configuration file generated by the 'sp' script.

   <num-sample>
   The number of local searches performed by this heuristic.

   <random-seed>
   A random number seed.

   <impact-file-representation>
   An integer that indicates how the impact file is stored internally:
   0 - sparse and 1 - dense

   <time-limit>
   A time limit (in seconds) for how long the heuristic should run.

   <solution-file>
   The name of the output file that contains the solutions found
   by this heuristic.
```

D.4 Description

The `sp` command runs `randomsample` to solve p-median sensor sensor placement problems with the GRASP heuristic. Currently, the following statistics are supported: mean, var (Value At Risk), tce (Tail-Conditional Expectation), and worst-case.

This command is intended to be only used by the **sp** script, which drives both heuristic and exact solvers.

D.5 Notes

- The randomsample heuristic currently does not support side constraints other than on the number of sensors. Side-constraints are supported by the **sideconstraints** (p. 47) executable.

E Executable scenarioAggr

E.1 Overview

The **scenarioAggr** executable takes an IMPACT file and produces an aggregated impact file.

E.2 Usage

```
scenarioAggr --numEvents=<num_incidents> <impact file>
```

E.3 Options

```
--numEvents=<number>
   The number of incidents that should be aggregated.
```

E.4 Description

The **scenarioAggr** executable reads an IMPACT file, finds similar incidents, combines them, and writes out another IMPACT file. The convention is to prepend the string "aggr" to the output.

The following files are generated during the execution of **scenarioAggr**, assuming that the input was named "network.impact":

- aggrnetwork.impact - the new **Impact File** (p. 33)
- aggrnetwork.impact.prob - the probabilities of the aggregated incidents. These are non-uniform, so any solver must recognize incident probabilities.

E.5 Notes

- Not all solvers in SPOT can perform optimization with aggregated IMPACT files. In particular, the heuristic GRASP solver does not currently support aggregation because it does not use incident probabilities. The Lagrangian and PICO solvers support incident aggregation. However, initial results suggest that although the number of incidents is reduced significantly, the number of impacts may not be, and solvers may not run much faster.

F Executable setupIPData

F.1 Overview

The **setupIPData** executable is used by the **sp** solver interface to setup an integer programming formulation for sensor placement.

F.2 Usage

```
setupIPData <sp-config-file>
```

F.3 Arguments

```
<sp-config-file>
Contains all of the information needed to setup a sensor placement
problem.
```

F.4 Description

The **setupIPData** executable is used by **sp** (p. 48) to setup an integer programming problem for sensor placement that can be solved using the GeneralSP IP model. The input file used by **setupIPData** is a **Sensor Placement Configuration File** (p. 35). The output of this executable is sent to standard out, and it is in a format that can be processed by **PICO** (p. 43) and the AMPL solver interface.

F.5 Notes

- This executable is not meant to be run interactively.
- The scalability of this solver has not been well-characterized for large datasets, even when using aggressive aggregation.

G Executable sideconstraints

G.1 Overview

The **sideconstraints** executable heuristically solves p-median formulations of the sensor placement problem where one or more side-constraints are specified. These side constraints are tight, meaning that any solution that violates the side constraints is considered infeasible.

G.2 Usage

```
   sideconstraints <sp-configuration-file> <num-samples> <random-seed>
<impact-file-representation> <time-limit> [<solution-file>]
```

G.3 Arguments

```
   <sp-configuration-file>
   The configuration file generated by the 'sp' script.

   <num-sample>
   The number of local searches performed by this heuristic.

   <random-seed>
   A random number seed.

   <impact-file-representation>
   An integer that indicates how the impact file is stored internally:
   0 - sparse and 1 - dense

   <time-limit>
   A time limit (in seconds) for how long the heuristic should run.

   <solution-file>
   The name of the output file that contains the solutions found
   by this heuristic.
```

G.4 Description

The `sp` command runs `sideconstraints` to solve p-median sensor placement problems that include side-constraints with the GRASP heuristic. Currently, the following statistics are supported: mean, var (Value At Risk), tce (Tail-Conditional Expectation), and worst-case.

G.5 Notes

None.

H Executable sp

H.1 Overview

The **sp** executable provides a common interface for sensor placement solvers in TEVA-SPOT.

H.2 Command-Line Help

```
Usage: sp [options]

Options:
  -h, --help            show this help message and exit
  -n NETWORK, --network=NETWORK
                        Name of network file
  --objective=OBJECTIVE
                        Objective names have the form: <goal>_<statistic>
                        ..The objective goals are:
                        ....cost   the cost of sensor placement
                        ....ec     extent of contamination
                        ....dec    detected extent of contamination
                        ....td     time to detection
                        ....dtd    detected time to detection
                        ....mc     mass consumed
                        ....dmc    detected mass consumed
                        ....nfd    number of failed detections
                        ....ns     the number of sensors
                        ....pe     population exposed
                        ....dpe    detected population exposed
                        ....pk     population killed
                        ....dpk    detected population killed
                        ....pd     population dosed
                        ....dpd    detected population dosed
                        ....vc     volume consumed
                        ....dvc    detected volume consumed
                        ..The objective statistics are:
                        ....mean   the mean impact
                        ....median the median impact
                        ....var    value-at-risk of impact distribution
                        ....tce    tail-conditioned expectation of imp dist
                        ....cvar   approximation to TCE used with IPs
                        ....worst  the worst impact
                        An objective name of the form <goal> is assumed to
                        refer to the objective <goal>_mean.  This option may
                        be listed more than once.
  -r DELAY, --responseTime=DELAY
                        This parameter indicates the number of minutes that
                        are needed to respond to the detection of a
                        contaminant.  As the response time increases, the
                        impact increases because the contaminant affects the
                        network for a greater length of time. Unit: minutes.
  -g GAMMA, --gamma=GAMMA
                        Specifies the fraction of the distribution of impacts
                        that will be used to compute the var, cvar and tce
                        measures.  Gamma is assumed to lie in the interval
                        (0,1].  It can be interpreted as specifying the
                        100*gamma percent of the worst contamination incidents
                        that are used for these calculations.  Default: .05
  --imperfect-scfile=SCFILE
                        Specifies the name of a file defining detection
                        probabilities for all sensor categories. Used with the
                        imperfect-sensor model.  Must be specified in
                        conjunction with the --imperfect-jcfile option.
```

```
--imperfect-jcfile=JCFILE
                      Specifies the name of a file defining a sensor
                      category for each network junction. Used with the
                      imperfect-sensor model. Must be specified in
                      conjunction with the --imperfect-scfile option.
--num=NUMSAMPLES, --numsamples=NUMSAMPLES
                      Specifies the number of candidate solutions generated
                      by the grasp heuristic. Defaults vary based on
                      statistic and sensor model formulation (perfect vs.
                      imperfect).
--grasp-representation=GRASP_REPRESENTATION
                      Specifies whether the grasp heuristic uses a sparse
                      matrix (0) or dense matrix (1) representation to store
                      the impact file contents.  The default is 1.
--impact-dir=IMPACT_DIRECTORY
                      Specifies the directory the contains impact files.  By
                      default the current directory is used.
--aggregation-threshold=AGGREGATION_THRESHOLD, --threshold=AGGREGATION_THRESHOLD
                      Specifies the value (as '<goal>,<value>') used to
                      aggregate 'similar' impacts.  This is used to reduce
                      the total size of the sensor placement formulation
                      (for large problems). The solution generated with non-
                      zero thresholds is not guaranteed to be globally
                      optimal.
--aggregation-percent=AGGREGATION_PERCENT, --percent=AGGREGATION_PERCENT
                      A '<goal>,<value>' pair where value is a double
                      between 0.0 and 1.0.  This is an alternate way to
                      compute the aggregation threshold. Over all
                      contamination incidents, we compute the maximum
                      difference d between the impact of the contamination
                      incident is not detected and the impact it is detected
                      at the earliest possible feasible location.  We set
                      the threshold to d * aggregation_percent.  If both
                      threshold and percent are set to valid values in the
                      command line, percent takes priority.
--aggregation-ratio=AGGREGATION_RATIO
                      A '<goal>,<value>' pair where value is a double
                      between 0.0 and 1.0.
--conserve-memory=MAXIMUM_IMPACTS
                      If location aggregation is chosen, and the original
                      impact files are very large, you can choose to process
                      them in a memory conserving mode.  For example "--
                      conserve_memory=10000" requests that while original
                      impact files are being processed into smaller
                      aggregated files, no more than 10000 impacts should be
                      read into memory at any one time.  Default is 10000
                      impacts.  Set to 0 to turn this off.
--distinguish-detection=DISTINGUISH_GOAL, --no-event-collapse=DISTINGUISH_GOAL
                      A goal for which aggregation should not allow events
                      to become trivial.That is, if the threshold is so
                      large that all locations, including the dummy, would
                      form a single superlocation, this forces the dummy to
                      be in a superlocation by itself.  Thus the sensor
                      placement will distinguish between detecting and not
                      detecting. This option can be listed multiple times,
                      to specify multiple goals.Note: the 'detected' impact
                      measures (e.g. dec, dvc) are always distinguished.
--disable-aggregation=DISABLE_AGGREGATION
                      Disable aggregation for this goal, even at value zero,
                      which would incur no error.  Each witness event will
                      be in a separate superlocation. This option can be
                      listed multiple times, to specify multiple goals. You
                      may list the goal 'all' to specify all goals.
--ub-constraint=UB_CONSTRAINT, --ub=UB_CONSTRAINT
                      This option specifies a constraint (<objective>,<ub-
                      value>) on the maximal value of an objective type.
                      This option can be repeated multiple times with
                      different objectives.
```

```
--baseline-constraint=BASELINE_CONSTRAINT, --baseline=BASELINE_CONSTRAINT
                      Baseline constraints are not currently supported.
--reduction-constraint=REDUCTION_CONSTRAINT, --reduction=REDUCTION_CONSTRAINT
                      Reduction constraints are not currently supported.
--costs=COST_FILE, --costs_ids=COST_FILE
                      This file contains costs for the installation of
                      sensors throughout the distribution network.  This
                      file contains id/cost pairs, and default costs can be
                      specified with the id: __default__.
--costs-indices=COST_INDEX_FILE
                      This file contains costs for the installation of
                      sensors throughout the distribution network.  This
                      file contains index/cost pairs, and default costs can
                      be specified with the index: -1.
--sensor-locations=LOCATIONS_FILE
                      This file contains information about whether network
                      ids are feasible for sensor placement, and whether a
                      sensor placement is fixed at a given location.
--solver=SOLVER       This option specifies the type of solver that is used
                      to find sensor placement(s).  The following solver
                      types are currently supported:
                      ..att_grasp  multistart local search heuristic (AT&T)
                      ..snl_grasp  TEVA-SPOT license-free grasp clone
                      ..lagrangian lagrangian relaxation heuristic solver
                      ..pico       mixed-integer programming solver (PICO)
                      ..glpk       mixed-integer programming solver (GLPK)
                      ..picoamp    MIP solver with AMPL
                      ..cplexamp   commercial MIP solver
                      The default solver is snl_grasp.
--solver-options=SOLVER_OPTIONS
                      This option contains solver-specific options for
                      controlling the sensor placement solver.  The options
                      are added to the solver command line.
--runtime=RUNTIME     Terminate the solver after the specified number of
                      wall clock minutes have elapsed.  By default, no limit
                      is placed on the runtime.  Some solvers can provide
                      their best solution so far at the point of
                      termination.
--notify=INTERVAL     Some solvers can output preliminary solutions while
                      they are running. This option supplies the interval in
                      minutes at which candidate solutions should be printed
                      out.
--compute-bound       Only compute a bound on the value of the optimal
                      solution.
--memmon              Summarize the maximum memory used by any of the
                      executables
--memcheck=MEMCHECKTARGET
                      This option indicates that valgrind should run on one
                      or more executables.
                      ..all               run on all executables
                      ..solver            run on the solver executable
                      ..setupIPData       run on setupIPData
                      ..preprocessImpacts run on preprocessImpacts
                      ..evalsensor        run on evalsensor
                      ..aggregateImpacts  run on aggregateImpacts  Output
                      will be written to memcheck.{name}.{pid} .
--tmp-file=TMP_FILE   Name of temporary file prefix used in this
                      computation. The default name is '<network-name>'.
-o OUTPUT_FILE, --output=OUTPUT_FILE
                      Name of the output file that contains the sensor
                      placement. The default name is '<network-
                      name>.sensors'.
--summary=SUMMARY_FILE
                      Name of the output file that contains summary
                      information about the sensor placement.
--format=FORMAT       Format of the summary information
--print-log           Print the solver output
--path=PATH           Add this path to the set of paths searched for
```

```
                      executables and IP models.
--amplcplexpath=AMPLCPLEXPATH
                      Look for ampl and cplexamp executables in this
                      directory. This defaults to a 'blank' path, which
                      implies that the user's system path is used.
--picopath=PICOPATH   Look for the PICO executable in this directory. This
                      defaults to the path used for executables specified
                      by. the --path option.
--glpkpath=GLPKPATH   Look for the GLPK executable in this directory. This
                      defaults to the path used for executables specified by
                      the --path option.
--ampl=AMPL           The name of the ampl executable (this defaults to
                      'ampl').
--ampldata=AMPLDATA   An auxillary AMPL data file.  This option is used when
                      integrating auxillary information into the AMPL IP
                      model.
--amplmodel=AMPLMODEL
                      An alternative AMPL model file.  This option is used
                      when applying a non-standard AMPL model for solving
                      sensor placement with an IP.
--seed=SEED           The value of a seed for the random number generator
                      used by the solver.  This can be used to ensure a
                      deterministic, repeatable output from the solver.
                      Should be >= 1.
--eval-all            This option specifies that all impact files found will
                      be used to evaluate the final solution(s).
--debug               List status messages while processing.
--gap=GAP             TODO gap help string.
--version             Print version information for the compiled executables
                      used by this command.
```

H.3 Description

The **sp** executable is a Python script that coordinates the execution of the SPOT sensor placement solvers. The **sp** options can flexibly specify the objective to be optimized, as well as constraints on performance/cost goals.

The **sp** script currently interfaces with integer programming (IP) solvers, GRASP heuristics, and a Lagrangian heuristic. The IP formulation can be used to find globally optimal solutions, the GRASP heuristic has proven effective at finding optimal solutions to a variety of large-scale applications, and the Lagrangian heuristic finds a near-optimal selection while computing a confidence bound.

The following files are generated during the execution of **sp**:

- <tmpfile>.config - the **Sensor Placement Configuration File** (p. 35)
- <tmpfile>.dat - an AMPL data file (if using an IP solver)
- <tmpfile>.mod - an AMPL script (if using an IP solver)
- <tmpfile>.log - a log file that captures the solver output
- <tmpfile>.sensors - a **Sensor Placement File** (p. 33) that contains the final solution

H.4 Notes

- The solvers provided by SPOT do not attempt to minimize the number of sensors that are used. This can sometimes lead to confusing behavior, especially for worst-case objectives where there may be different solutions with different numbers of sensors. For small problems, the PICO solver can be used to solve an auxilliary problem, where the number of sensors is minimized subject to the performance value that is found when minimizing impact.

- The heuristic solvers do not currently support the "median" performance measure.
- The IP solvers do not currently support median, var, or tce performance measures.
- The aggregation threshold does not currently impact the problem formulation used by the GRASP heuristic.
- This solver interface assumes that event likelihoods are uniform. The format for specifying non-uniform likelihoods remains unresolved.
- Numerical issues have been observed when solving with the PICO solver in some cases. These usually result in a message that indicates that the solver failed.
- The gamma parameter cannot be varied with snl_grasp or att_grasp.
- The snl_grasp and att_grasp solvers cannot effectively minimize the worst-case objective when the problem is constrained.
- The "ub" option to sp is a misnomer when the Lagrangian solver is selected. The side constraints are really goal constraints, and therefore the values specified are not true upper bounds. However, we have decided to keep a consistent usage for both side and goal constraints rather than introducing a new option.
- The Lagrangian heuristic can now be used with "witness aggregated" problems as the PICO solver can. This cuts down the required memory in a dramatic way. For example, a problem that caused the Lagrangian to use 1.8 GB of RAM and run in 20 minutes when unaggregated, was solved with only 27MB of RAM in 20 seconds when aggregated. There is a disadvantage, though. The actual quality of the witness-aggregated solution of the Lagrangian solver can be 25 percent (or more) worse than the unaggregated solution. This could be improved in the future.
- The sp executable currently defaults to invoke witness aggregation when the Lagrangian solver is selected. If you want to turn this feature off, you must use the disable-aggregation option. To disable aggregation of all objectives, use the option disable-aggregation=all, as in the example above.

I Executable tevasim

I.1 Overview

The **tevasim** executable uses EPANET to perform an ensemble of contaminant transport simulations.

I.2 Command-Line Help

```
Usage: tevasim [options] <epanet-input-file> <epanet-output-file>
A utility for running an ensemble of water quality simulations, whose results
are stored in a TSO file.

options:

  -h, --help                  Display usage information
      --tsg                   The TSG file used to specify the injection
                              incidents.
      --tsi                   The TSI file used to specify the injection
                              incidents.
  -v, --tso-version           The version of the TSO format that is generated.
      --tso                   The TSO output file.
      --version               Display version information

arguments:

  epanet-input-file:  EPANET network file.

  epanet-output-file:  Output file generated by EPANET.

The tevasim command is used to simulate contamination incidents. This command
uses EPANET to perform an ensemble of contaminant transport simulations,
defined by a TSG File. The following files are generated during the execution
of tevasim:
  - a binary TSO file that contains the contamination transport data,
  - a binary SDX file that provides an index into the TSO File, and
  - an output file that provides a textual summary of the EPANET simulations.

Note that options like ‘tso’ can be specified with the syntax ‘--tso file.tso’
or ‘--tso=file.tso’.
```

J Executable tso2Impact

J.1 Overview

The **tso2Impact** executable generates one or more impact files from a TSO file.

J.2 Command-Line Help

```
Usage: tso2Impact [options] <output-prefix> <tso-directory-or-files>
                            <tai-directory-or-files>
An application that reads a TSO file (and associated TAI file if health
impacts are to be computed) and creates one or more impact files that are used
to formulate sensor placement problems.

options:

      --dec                   If this option is specified, an impact file will
                              be generated for the 'detected extent of
                              contamination' measure.
      --detectionConfidence   The number of sensors that must detect an event
                              before the impacts are calculated.  Normally
                              this is 1 sensor.
  -d, --detectionLimit        A list of thresholds needed to perform detection
                              with a sensor.  There must be one threshold for
                              each .tso file. The units of these detection
                              limits depend on the units of the contaminant
                              simulated for each TSO file (e.g. number of
                              cells of a biological agent).
      --dmc                   If this option is specified, an impact file will
                              be generated for the 'detected mass consumed'
                              measure.
      --dpd                   If this option is specified, an impact file wll
                              be generated for the 'detected population dosed'
                              measure.  This is an intensive measure to
                              compute.
      --dpe                   If this option is specified, an impact file wll
                              be generated for the 'detected population
                              exposed' measure.  This is an intensive measure
                              to compute.
      --dpk                   If this option is specified, an impact file wll
                              be generated for the 'detected population
                              killed' measure.  This is an intensive measure
                              to compute.
      --dtd                   If this option is specified, an impact file will
                              be generated for the 'detected
                              time-to-detection' measure.
      --dvc                   If this option is specified, an impact file will
                              be generated for the 'detected volume consumed'
                              measure.
      --ec                    If this option is specified, an impact file will
                              be generated for the 'extent of contamination'
                              measure.
      --epanetin              This is used for TSO file versions less than
                              6.0, when computation of the 'ec' objective is
                              specified.  Pipelengths are extracted from the
                              EPANET input file.
  -h, --help                  Display usage information
      --mc                    If this option is specified, an impact file will
                              be generated for the 'mass consumed' measure.
      --nfd                   If this option is specified, an impact file will
                              be generated for the 'number-of-failed
                              detections' measure.
```

```
      --pd                    If this option is specified, an impact file wll
                              be generated for the 'population dosed' measure.
                              This is an intensive measure to compute.
      --pe                    If this option is specified, an impact file wll
                              be generated for the 'population exposed'
                              measure.  This is an intensive measure to
                              compute.
      --pk                    If this option is specified, an impact file wll
                              be generated for the 'population killed'
                              measure.  This is an intensive measure to
                              compute.
  -r, --responseTime          This option indicates the number of minutes that
                              are needed to respond to the detection of a
                              continant.  As the response time increases, the
                              impact increases because the contaminant affects
                              the network for a greater length of time. Unit:
                              minutes.
      --td                    If this option is specified, an impact file will
                              be generated for the 'time-to-detection'
                              measure.
      --tsoPattern            This string specifies a regular expression for
                              all input TSO files when the files are stored in
                              a directory.
      --vc                    If this option is specified, an impact file will
                              be generated for the 'volume consumed' measure.
      --version               Display version information

arguments:

  output-prefix:  The prefix used for all files generated by tso2Impact.

  tso-directory-or-file:  This argument indicates either a TSO file or a
      directory name for TSO files.  If the later, then the filenames must be
      specified with the --tsoPattern option.

  tai-directory-or-file:  This argument indicates a TAI file name.  The TAI
      input file is a threat_assess input that specifies parameters like
      dosage, response, lethality, etc.  There should be one TAI file for each
      TSO file.

Note that options like 'responseTime' can be specified with the syntax
'--responseTime 10.0' or '--responseTime=10.0'.
```

J.3 Description

The **tso2Impact** executable generates impact files that are used for sensor placement. This executable processes a **TSO File** (p. 32), which summarizes the result of an EPANET computation. The following files are generated during the execution of **tso2Impact:**

- <output-prefix>.nodemap - a **Node File** (p. 34).
- <output-prefix>.scenariomap - a **Scenario File** (p. 34).
- <output-prefix>_<impact-type>.impact - an **Impact File** (p. 33) for a given impact.

J.4 Notes

- The '–tsoPattern' option allows a set of TSO files to be specified without explicitly listing all of them on the command-line. The user specifies a regular expression, and all files that match that expression are included in the analysis.

K Executable ufl

K.1 Overview

The **ufl** executable heuristically solves p-median formulations of the sensor placement problem while also computing a valid lower bound on the best possible sensor placement value.

K.2 Usage

```
   ufl <sp-configuration-file> <p> [--gap=<fraction>]
[<goal_constraint_data_file> <upper_bound>]*
```

K.3 Options

```
   --gap=<fraction>
       This option tells the solver to stop when the solution is
       within a certain percentage of optimal.  Let \b icost be the current
       best integer solution found and \b lb be the current lower bound.
       The solver will stop with <b> (icost - lb)/lb </b> is less that the
       gap.  For example, if the gap is 0.1, then the solver will stop when
       it has a solution that is within 10 percent of optimality.
```

K.4 Arguments

```
   <sp-configuration-file>
       A LAG file that defines impacts for the objective.

   <p>
       The number of sensors.

   <goal_constraint_data_file>
       A LAG file that defines impacts for a side-constraint.

   <upper_bound>
       The upper bound for this side constraint.
```

K.5 Description

"ufl" stands for "uncapacitated facility location," and this code is a Sandia-modified version of the combination of Lagrangian relaxation and the "Volume Algorithm" that is found in the open-source "COIN" repository (that the PICO solver uses).

The **sp** executable automatically generates **ufl** commands, including those with goal constraints. The user specifies the number of sensors, and **sp** passes to **ufl** one more than this number. The Lagrangian heuristic implemented in **ufl** then places the correct number of sensors, and one "dummy" sensor that catches all undetected events.

Note that the **ufl** command uses **LAG File** (p. 34) inputs, which are a modified format of impact files. These files are generated by the **tso2Impact** (p. 54) executable.

As of teva-spot-1.2, **ufl** handles "goal constraints." For example, we may minimize the contaminant mass consumed subject to the goal of limiting the extent of contamination in pipe feet to a constant such as 15,000. This is different from specifying a side constraint for the "sideconstraints" local search executable. The latter will reject any solution in which the extent of contamination is greater than 15,000, even if it is

only 15,001. Many goal constraints may be provided simultaneously, and the Lagrangian solver will attempt to find a solution that honors those constraints. It will report one that has a good combination of primary objective value and small violations of the goals.

This technology is young, and experience shows that user attempts to make the goal constraints too tight can confuse the solver. We offer the following guidance to avoid this problem. Suppose that we wish to use the Lagrangian heuristic to find a good solution that minimizes the average contaminant mass consumed subject to utility guidelines on the average extent of contamination, and also the average volume of contaminated water consumed.

```
1.  Using a solver of choice for the particular problem, find single-
    objective optimal values for each objective.
2.  Using evalsensor, evaluate the single-objective sensor placements
    against each of the other objectives.  The result is a matrix of
    objective values.
3.  Determine goal constraints for the secondary objectives by selecting
    a value between the optimal single-objective value for that secondary
    objective, and its value under the sensor placement obtained by solving
    the single-objective problem for the primary objective.
```

For example, for a real test problem, minimizing the average contaminant mass consumed yielded an objective value of 638,344 units. Taking the sensor placement obtained from that solve, we found that the average extent of contamination was 78,037 feet, and the average volume of contaminated water consumed was 282,689 units.

Solving individually for these objectives, we found that the optimal solutions for extent of contamination and volume consumed were 40,867 and 217,001, respectively. From this information, we decided to apply goal constraints of 45,000 feet for the extent of contamination, and 250,000 units for the volume consumed.

Minimizing the mass consumed with these two goal constraints, the Lagrangian heuristic found a new sensor placement that incurred objective values of 678,175 units for mass consumed, 49,016 feet for the extent of contamination, and 256,615 units for volume consumed. Note that neither goal was strictly met, but each goal helped improve its related objective value.

We now compare this technology to the side-constrained local search heuristic (the "sideconstraints" executable). Each heuristic has advantages and disadvantages. The goal-constrained Lagrangian solver can handle an arbitrary number of goal constraints, producing a solution that is well balanced, as above. When we attempt to reproduce the results above using the "sideconstraints" executable, which is currently limited to only one side constraint, we see the untreated objective suffer. For example, with the same setup as above, and a side constraint of 50,000 feet for average extent of contamination, the sideconstraints heuristic produces a solution with an expected mass consumed of 670,399 units, an expected extent of contamination of 49,827 feet, and an expected volume consumed of 326,943. We see a similar type of result with a single side constraint on the volume consumed (the extent of contamination increases substantially). The sideconstraints code could be extended to handle multiple side constraints, of course, but the neighborhood search might have difficulty finding feasible solutions. Since Lagrangian relaxation is a global technique and slightly infeasible solutions are permitted, we are more likely to find a good trade-off.

However, the Lagrangian heuristic has disadvantages as well. If a particular goal constraint is set too tightly, the solution can degenerate such that all of the objectives get substantially worse. We do not understand this phenomenon well yet, and further research into the algorithm itself may be necessary to make this technology generally usable. For now, it is sometimes necessary to manipulate the values of the goal constraints manually in order to find a good solution.

K.6 Notes

None.

L LGPL License

The TEVA-SPOT Toolkit software is distributed under the terms of the GNU Lesser General Public License (LGPL) (see disclaimer at the beginning of this work). For your reference, the GNU LGPL and the GNU General Public License (GPL), of which the LGPL is a derivative, are included in this appendix. The GNU LGPL is contained in `admin/LICENSE.lgpl`. The GNU GPL is contained in the file `admin/LICENSE.gpl`.

L.1 GNU Lesser General Public License

```
   GNU LESSER GENERAL PUBLIC LICENSE
                       Version 3, 29 June 2007

 Copyright (C) 2007 Free Software Foundation, Inc. <http://fsf.org/>
 Everyone is permitted to copy and distribute verbatim copies
 of this license document, but changing it is not allowed.


  This version of the GNU Lesser General Public License incorporates
the terms and conditions of version 3 of the GNU General Public
License, supplemented by the additional permissions listed below.

  0. Additional Definitions.

  As used herein, "this License" refers to version 3 of the GNU Lesser
General Public License, and the "GNU GPL" refers to version 3 of the GNU
General Public License.

  "The Library" refers to a covered work governed by this License,
other than an Application or a Combined Work as defined below.

  An "Application" is any work that makes use of an interface provided
by the Library, but which is not otherwise based on the Library.
Defining a subclass of a class defined by the Library is deemed a mode
of using an interface provided by the Library.

  A "Combined Work" is a work produced by combining or linking an
Application with the Library.  The particular version of the Library
with which the Combined Work was made is also called the "Linked
Version".

  The "Minimal Corresponding Source" for a Combined Work means the
Corresponding Source for the Combined Work, excluding any source code
for portions of the Combined Work that, considered in isolation, are
based on the Application, and not on the Linked Version.

  The "Corresponding Application Code" for a Combined Work means the
object code and/or source code for the Application, including any data
and utility programs needed for reproducing the Combined Work from the
Application, but excluding the System Libraries of the Combined Work.

  1. Exception to Section 3 of the GNU GPL.

  You may convey a covered work under sections 3 and 4 of this License
without being bound by section 3 of the GNU GPL.

  2. Conveying Modified Versions.

  If you modify a copy of the Library, and, in your modifications, a
facility refers to a function or data to be supplied by an Application
that uses the facility (other than as an argument passed when the
facility is invoked), then you may convey a copy of the modified
version:
```

a) under this License, provided that you make a good faith effort to
ensure that, in the event an Application does not supply the
function or data, the facility still operates, and performs
whatever part of its purpose remains meaningful, or

b) under the GNU GPL, with none of the additional permissions of
this License applicable to that copy.

3. Object Code Incorporating Material from Library Header Files.

The object code form of an Application may incorporate material from a header file that is part of the Library. You may convey such object code under terms of your choice, provided that, if the incorporated material is not limited to numerical parameters, data structure layouts and accessors, or small macros, inline functions and templates (ten or fewer lines in length), you do both of the following:

a) Give prominent notice with each copy of the object code that the
Library is used in it and that the Library and its use are
covered by this License.

b) Accompany the object code with a copy of the GNU GPL and this license
document.

4. Combined Works.

You may convey a Combined Work under terms of your choice that, taken together, effectively do not restrict modification of the portions of the Library contained in the Combined Work and reverse engineering for debugging such modifications, if you also do each of the following:

a) Give prominent notice with each copy of the Combined Work that
the Library is used in it and that the Library and its use are
covered by this License.

b) Accompany the Combined Work with a copy of the GNU GPL and this license
document.

c) For a Combined Work that displays copyright notices during
execution, include the copyright notice for the Library among
these notices, as well as a reference directing the user to the
copies of the GNU GPL and this license document.

d) Do one of the following:

0) Convey the Minimal Corresponding Source under the terms of this
License, and the Corresponding Application Code in a form
suitable for, and under terms that permit, the user to
recombine or relink the Application with a modified version of
the Linked Version to produce a modified Combined Work, in the
manner specified by section 6 of the GNU GPL for conveying
Corresponding Source.

1) Use a suitable shared library mechanism for linking with the
Library. A suitable mechanism is one that (a) uses at run time
a copy of the Library already present on the user's computer
system, and (b) will operate properly with a modified version
of the Library that is interface-compatible with the Linked
Version.

e) Provide Installation Information, but only if you would otherwise
be required to provide such information under section 6 of the
GNU GPL, and only to the extent that such information is
necessary to install and execute a modified version of the
Combined Work produced by recombining or relinking the
Application with a modified version of the Linked Version. (If

you use option 4d0, the Installation Information must accompany
the Minimal Corresponding Source and Corresponding Application
Code. If you use option 4d1, you must provide the Installation
Information in the manner specified by section 6 of the GNU GPL
for conveying Corresponding Source.)

5. Combined Libraries.

You may place library facilities that are a work based on the Library side by side in a single library together with other library facilities that are not Applications and are not covered by this License, and convey such a combined library under terms of your choice, if you do both of the following:

a) Accompany the combined library with a copy of the same work based on the Library, uncombined with any other library facilities, conveyed under the terms of this License.

b) Give prominent notice with the combined library that part of it is a work based on the Library, and explaining where to find the accompanying uncombined form of the same work.

6. Revised Versions of the GNU Lesser General Public License.

The Free Software Foundation may publish revised and/or new versions of the GNU Lesser General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Library as you received it specifies that a certain numbered version of the GNU Lesser General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that published version or of any later version published by the Free Software Foundation. If the Library as you received it does not specify a version number of the GNU Lesser General Public License, you may choose any version of the GNU Lesser General Public License ever published by the Free Software Foundation.

If the Library as you received it specifies that a proxy can decide whether future versions of the GNU Lesser General Public License shall apply, that proxy's public statement of acceptance of any version is permanent authorization for you to choose that version for the Library.